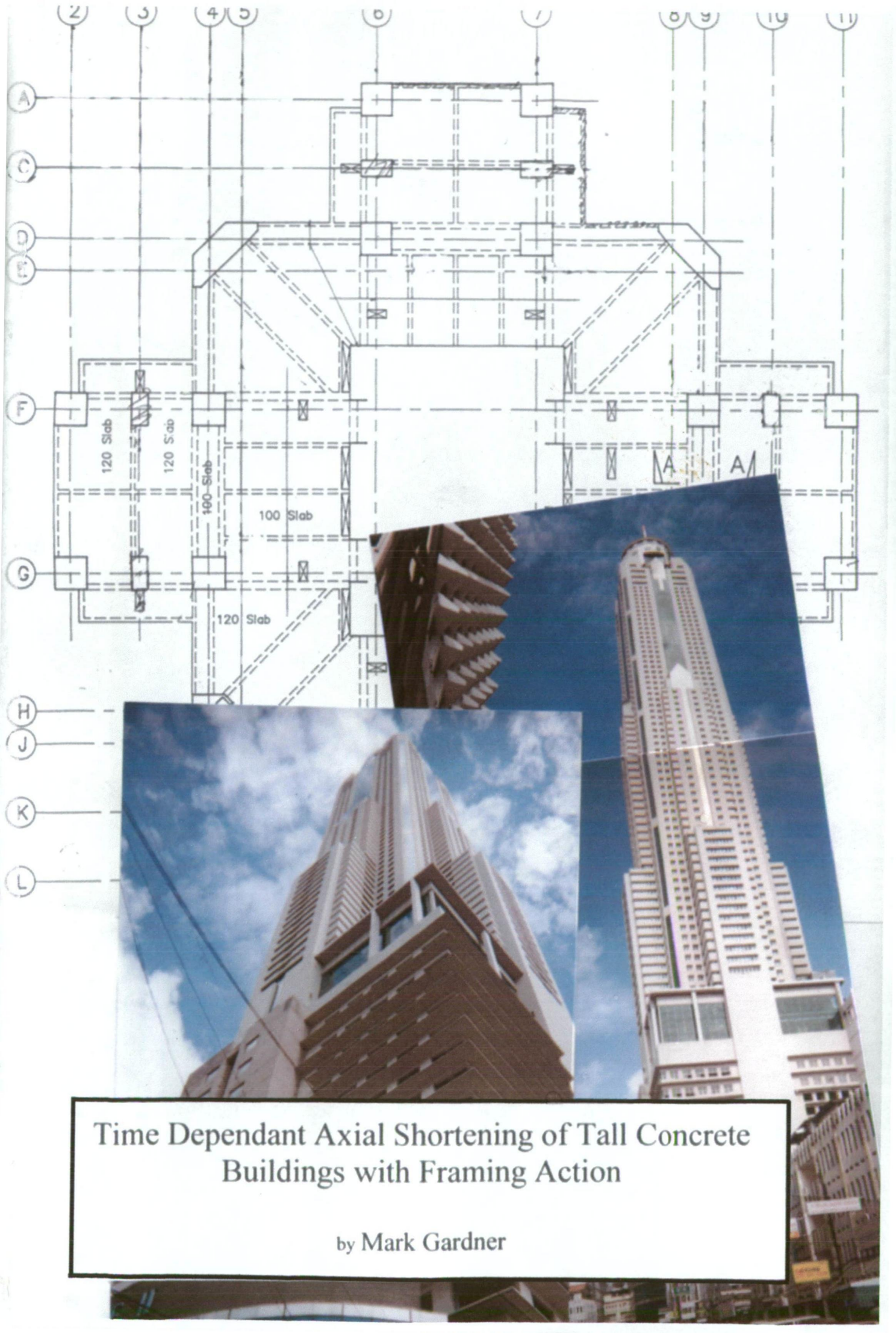


Time Dependant Axial Shortening of Tall Concrete Buildings with Framing Action

by Mark Gardner

On the Time Dependent Axial Shortening of Tall Concrete Buildings with Framing Action

Section 1

by Mark Gardner BE(hons) MIEAust CPEng

Submitted in fulfilment of the
requirements for the degree of Doctor of Philosophy

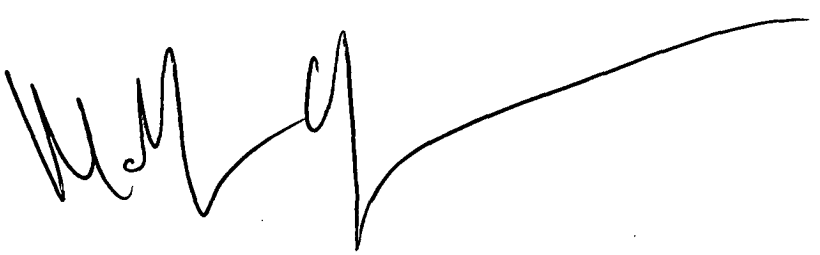
University of Tasmania
March 2004

ORIGINALITY

This thesis contains no material that has been previously accepted by the University of Tasmania, or any other institution unless it has been duly acknowledged in the thesis and the relevant author cited. To the best of the authors knowledge it contains only new researched material not yet written or published by any other person except where due acknowledgement is made.

Signed

Mark Gardner.



AUTHORITY OF ACCESS

ABSTRACT

Long term axial shortening in a tall concrete structure is a second-order effect that is of some impediment to the serviceability, and in some cases, structural integrity of the building. The research here involves analysis of axial shortening elements that interact through framing action. In many tall concrete buildings framing is inherent as columns, beams and floors are quite often connected with rigid connections. The mechanics of differential axial shortening and framing action due to rigid connections causes load sharing between vertical elements that fundamentally affects the degree of both absolute and differential axial shortening. Evaluation of axial shortening by analysis of a structural system consisting of discrete elements over-predicts the level of axial shortening when framing is present.

A methodology to calculate axial shortening, accounting for load sharing of axial loads for tall concrete buildings, is developed. The application of this methodology to the construction of a tall concrete building allowing for the building cycle is presented. A program is written that incorporates both the methodology and its application to a tall building, using the ACI concrete creep and shrinkage models. Correlation of the program is made against an existing previously tested program that does not allow for framing action. A comparison is made between the two sets of data for an actual 85 storey building. From this it is concluded that;

i) Framing action reduces the differential axial shortening between two elements
ii) The core generally experiences less change to axial shortening than columns
iii) The effect of framing cannot be estimated by a relative percentage adjustment applied uniformly to each storey.

ACKNOWLEDGMENTS

This thesis is dedicated to my mother Hazel and father Alan and to my wife Annie Gardner for all their support during the many hours dedicated to its final publication.

Additional thanks are made to the following people:

Allan Beasley	For support and guidance. As well as constant advice and detailed editing, ensuring I never lost sight of the final product.
Glenn Lewis	For much valued advice tutoring me in Java programming, and providing assistance refining the program.
Brian Cousins	For assistance with Excel® programming.
Edmund Melerski	For assistance with structural matrix analysis.
Simon Weatherspoon	For his mathematical input.
Jenny Gardner	For her valuable assistance editing.
Phisit Limtrakun	For being a good friend.

and to all the others that helped in any way.

NOTATION

α	constant
α	instantaneous axial strain at time (s)
$A(t,s)$	discrete elastic and creep component of axial shortening
A_c	area of concrete
A_s	area of steel
$[A]_2$	is the identity matrix for a 2 x 2 matrix
β	constant
$B(t)$	discrete shrinkage component of axial shortening
$C(t)$	discrete reinforcing component of axial shortening
$\delta_3(t)$	differential shortening ($d_1(t) - d_2(t)$) [notation adopted by Warner (1975)]
$\delta_c(t,s)$	creep axial shortening component
$\delta_e(s)$	elastic axial shortening component
$\delta_{j,1}$	incremental time-varying axial shortening of column 1 at the j^{th} storey
$\delta_r(t)$	reinforcing stiffening axial shortening component
$\delta_s(t)$	shrinkage axial shortening component
$\delta_{tot}(t)$	total time dependant axial shortening of the column at time t
$\{\delta_n\}_2$	2 x 1 column of associated axial shortening values at the n^{th} storey
$\delta\delta_{j,1}$	sum of the incremental time-varying axial shortening of column 1 from the first storey to the j^{th} storey
$\Delta^D_{j,1\|2}$	differential axial shortening value of the column pair 1 and 2 at the j^{th} storey
$\Delta\varepsilon''(t)$	combination of creep and shrinkage strains
$\Delta_{j,1}$	total time-varying axial shortening values of column 1 at the j^{th} storey
$\Delta\sigma(t)$	difference in stress over time after application of load
$\varepsilon(t)$	total strain at (t) days from casting of concrete
$\varepsilon(t)$	time dependant total strain in the member
$E''(t,s)$	age adjusted effective modulus
$E_c(s)$	modulus for the concrete section at the application of load
E_s	modulus of elasticity for steel
$\varepsilon_c/\varepsilon_{c\infty}$	creep strain to max creep strain ratio [notation adopted by Hansen (1966)]

$\varepsilon_s/\varepsilon_{s\infty}$ shrinkage strain to max shrinkage strain ratio [notation adopted by Hansen [26]]

$\varepsilon_{sh}(t)$ total shrinkage strain at (t) days from casting of concrete

$\varepsilon_{sh}(u)$ ultimate shrinkage

ε_{sh} mean total shrinkage strain

$\varepsilon_{st}(t)$ time dependant total steel strain

ϕ (t,s) creep coefficient, where (s) is the time at load application and t > s

ϕ_u ultimate creep coefficient

$F_{j,1}$ framing force in column 1 at the j^{th} storey resulting from the transfer of shear forces from the framing member between columns 1 and 2.

I_g gross moment of inertia

k_j stiffness relationship between F_j and Δ^D_j and is assumed linear for values of F_j that cause stresses within 50% of the ultimate stress capacity of the concrete element.

$k_{j,1|2}$ stiffness multiplier at column 1 for framed connections between columns 1 and 2.

L total effective length of the column with constant axial load P^I

n(s) modular ration at time of load application

n'(t,s) effective modular ratio

n''(t,s) age-adjusted effective modular ratio

N_j node at the jth storey

p ratio of cross-sectional areas of steel to concrete $= \frac{A_s}{A_c}$

$P_1(t)$ internal axial force in Member 1 [notation adopted by Warner (1975)]

$P_2(t)$ internal axial force in Member 2 [notation adopted by Warner (1975)]

P_A constant applied load at Member 1 [notation adopted by Warner (1975)]

P_B constant applied load at Member 2 [notation adopted by Warner (1975)]

$P^E_{j,1}$ external force at the j^{th} storey nodes for columns 1

$P^I(s)$ applied load at time (s), assumed uniform across entire section

$P^I_{j,1}$ resultant internal compression force in column 1 supporting the j^{th} storey, from the equilibrium of nodal forces at the j^{th} storey.

$\{P^I_n\}_2$ 2 x 1 vector of internal forces at the n^{th} storey

$\{P^E_n\}_2$ 2 x 1 vector of external forces at the n^{th} storey

s age at loading (in days)

S_i stage (i) of (n) possible stages

$\sigma_c(s)$	initial stress at time of loading (s)
$\sigma_c(t)$	total stress in the concrete at time (t)
$\sigma_s(t)$	total stress in the steel at time (t)
t	total age of element (in days)
t_0	age at onset of drying (in days)
t_d	age of concrete from onset of drying (in days)
v/s	volume to surface ratio
$\chi(t,s)$	aging coefficient
x, y, z	global coordinate system (length, height, depth)

CONTENTS

Chapter 3

FRAMING

Appendix

Chapter 4

APPLICATION OF FRAMING MATRICES

Appendix

Chapter 5

RESULTS OF ASCA AXIAL SHORTENING ANALYSIS

Chapter 1

INTRODUCTION

1.1 General

Presently skyscrapers are synonymous with modern cities throughout Australia and the rest of the world. Although some are composite structures, many skyscrapers in Australia are fully reinforced concrete (Bursle 2003). Whilst the technology to build skyscrapers or tall concrete buildings (TCBs) is currently available, a number of factors including differential axial shortening of vertical elements (columns and cores) still requires further research.

This thesis endeavours to accurately determine the differential axial shortening in tall structures with specific new research directed at concrete structures which experience framing action between the vertical elements.

Presently the development of models to determine axial shortening in TCBs revolve around axial shortening analysis of isolated elements. Attempts have been made to account for framing action, however, currently there is no definitive approach or procedure that aims to evaluate differential long term shortening which accounts for framing action.

Controlling differential axial shortening can be best achieved at the design stage by ensuring that framing connections between beams and columns/cores are rigid (Placsek 2003). This allows loads to be distributed between these elements reducing differential shortening (Placsek 2003). The distribution of loads and resulting deformations need to be evaluated to enable accurate member design for both strength and serviceability.

1.2 Scope

This thesis develops a rational and rigorous method for calculating differential axial shortening of *framed* vertical elements in TCBs. The topic is of significant importance to designers as differential movements can cause both large internal stresses as well as alignment problems in floors, lifts, services and facades.

Current constitutive models for concrete are investigated and the most up to date of these is used to develop axial shortening computations for isolated vertical elements. The work by Beasley (1987) and Koutsoukis (1995-97) is invaluable in assisting to isolate and comprehend some of the more accurate models already developed. An understanding of this area is essential before further progress can be made. A working model for calculating creep and shrinkage for a TCB is derived from existing models in such a way that it can be used to determine axial shortening for a framed structure. From this, the model is developed to calculate axial shortening in framed structures.

The author has developed a computer program that is capable of predicting axial shortening values allowing for the effects of framing for a building of common geometry. The model utilises recognised constitutive models to calculate creep and shrinkage (ACI 1986). This thesis is directed primarily at solving axial shortening with framing action rather than the accuracy of the constitutive model. The model was chosen to best allow accurate comparison with an existing axial shortening program. The existing program developed by Beasley (1987) has been utilised in the design of several international buildings in conjunction with prominent structural engineering firms in Australia. It has also been calibrated with survey data of axial shortening values from the Bayoke Tower II in Thailand. These calibrated results allow the new computer program developed by this research to be calibrated in a similar fashion.

1.3 Thesis Outline

The thesis is divided into two sections. Section 1 describes all theoretical aspects of the topic. Section 2 contains the computer program written to calculate axial shortening values.

Section 1 is organised into seven chapters, each summarised below;

Chapter 1 introduces the topic and the concept of axial shortening. Definitions of axial shortening and framing are given. The scope of the research and a literature review are presented.

Chapter 2 discusses creep and shrinkage of concrete in detail. Both are defined, including the time dependant nature of each. The important principle of superposition is also discussed with mention of its relevance to axial shortening calculations taking into account framing action. The current concrete models used to predict creep and shrinkage and the model used for a reinforced concrete column are detailed. A creep and shrinkage model for the reinforced concrete element that is relevant to a framed concrete structure is developed.

Chapter 3 considers common rigid connections in modern multi-storey structures. A structured method for incorporating these rigid connections into the developed creep and shrinkage model for a reinforced concrete element is formulated . A matrix method, for calculation by computer of axial shortening, is developed to enable the calculation of a structure with multiple connections on multiple levels.

Chapter 4 considers the building process in detail with specific respect to the introduction of framing during the stages of construction. The matrix method for calculation of axial deformations is modified at all the discrete stages that impact on creep and shrinkage analysis. A numerical solution for a small *framed* structure is presented. The developed method is tested with data for a larger more complex

structure. The results are produced by a software package, ASCA, written by the author to perform the necessary calculations involved in the developed matrix method.

Chapter 5 discusses output from ASCA. Two data sets from the Bayoke Tower II in Thailand are investigated with framing action introduced. The results of the program are compared with those of a similar program, COLECS, that does not allow for framing action. Chapter 5 discusses how it is possible to calibrate ASCA against COLECS to determine the accuracy of the new proposed method used to calculate axial shortening with framing action.

Chapter 6 describes the development of ASCA which is used to perform the calculations involved with axial shortening modelling. The language, development environment, input and output aspects of the program are discussed in full. Although the logical sequence of the research undertaken suggests that this chapter should precede the results chapter, the flow of development of the theory is improved by keeping this format.

Chapter 7 makes a number of conclusions. These include the accuracy of results obtained, the relative usefulness of the software developed and recommended future work in this area.

Figure 1.1 Bayoke Tower II. Thailand.

1.4 Axial Shortening Implications

Axial shortening is a secondary design effect, in fact so secondary that as little as 30 years ago it was not even considered in the design of many tall structures (Fintel et al 1984).

Axial shortening occurs due to the deformations in vertical elements resulting from loading and concrete shrinkage. These deformations vary greatly for different materials and can continue to develop indefinitely throughout the life of a structure. Whilst concrete offers the designer many advantages construction wise, TCBs suffer from these deformations, and hence load redistribution over time.

If axial deformations are developed evenly across a structure, each level will deform vertically by the same amount and at the same rate, so that associated problems with the deformations are less apparent. Unfortunately this is not usually the case and it is quite common for spandrel columns to shorten differentially to adjacent structural cores by up to 150 mm (Fintel 1986). Measurement has shown that differential axial shortening can be up to 0.1% of the height of a structure (Fintel et al 1984). This starts to become more of a problem when this is extrapolated to an 80 storey building (300 metres) that may experience shortening of 300mm (Fintel et al 1984). Such a degree of axial shortening produces problems with ventilation pipes, water and waste pipes, heating systems and lifts. Exterior facades can also be stressed far more than the designer intended by differential shortening. Additionally, this can result in major differential shortening where floors develop a camber over time, which reduces their functionality.

Furthermore, whilst steel undergoes an initial axial shortening with no further axial shortening with time, concrete shortens less initially, but then continues to suffer axial shortening indefinitely to values 2.5 to 3 times the initial deflection (Warner 1975). This can cause large design problems when steel is used in conjunction with concrete, typically in lift shafts, where guide rails may need to be physically shortened over time to compensate.

For these reasons the designers of TCBs need to know the actual expected differential shortening of vertical elements at various stages in time. It may be necessary to build pre-cambers[1] into floors or even redesign vertical elements inducing artificial loads [2].

1.5 Framing

There are currently few models available for calculating axial shortening of vertical elements in multi-storey buildings. Researchers have been able to show that without framing considerations taken into account, current models all over-predict shortening (Bakoss et al 1984). Rather than simply eluding to the fact that the models all over-predict shortening, it is far more useful to determine the degree of over-prediction and in what areas. Research is needed on the current construction methods for concrete structures and how current models and approaches to axial framing calculations can be improved to reflect framing interactions in such structures. Consequently this project answers the following three questions;

i) What types of connections are used today in multi-storey concrete structures ?
ii) What are the degrees of framing of these connections ?
iii) How can we best model these connections ?

The above questions are fundamental in shaping the approach taken towards generating a model and developing the input procedure for the program. The results of the research are discussed further in Section 3.3.1 *Connections* and Section 4.2 *Building Cycle.*

[1] Pre-camber refers to the practice of constructing floors out of level at the construction stage so that at a time in the future the floors will level out as axial shortening develops

1.6 Literature Review

Two areas of research into concrete structures are relevant to the development of a model to calculate axial shortening of framed structures. The first is that of creep and shrinkage models for concrete and the principle of superposition. The second area is in the development of the design and analysis of multi-storey structures. These developments are presented in chronological order, highlighting the significant steps.

1.6.1 Studies of major importance on creep and shrinkage

In the late 1920s, concrete research began to focus on the long-term creep and shrinkage characteristics of concrete, which in this thesis are referred to as secondary effects. One of the first scientists to publish a significant study on the secondary effects of concrete was Faber (1927). In his work Faber referred to creep as the plastic yield of concrete[3]. He published results of tests carried out on four concrete beams reinforced in tension, all centrally loaded by equally increasing loads with steel stresses remaining below the yield point.

In a second experiment, shrinkage tests were made on two concrete beams, however this time the beams were equally reinforced at the top and bottom.

A number of important observations were derived from these tests. Firstly, as the beams continued to deflect with time, the largest axial movement was shown to be on the compression side rather than tension. Secondly, the deflections of all four beams did not increase equally even though the loading was applied in equal increments. Thirdly, the measured shortening of the compression flange with time was more than that for basic shrinkage of the same section[4]. From this, Faber developed his theory on plastic yield.

---

[2] It is not uncommon to provide vertical stressing of cores. This is not only to increase stiffness but also to enable control of axial shortening

[3] This is most likely a term borrowed from steel behaviour

[4] Basic shrinkage refers to a sample left to shrink undisturbed

From these early experiments, it was concluded that concrete undergoes a plastic yielding under axial load. Faber formulated a value for the modulus of elasticity for concrete. This was taken as simply the average slope of the stress/strain curve over a short period (Faber 1927).

In 1937, Davis and Brown published documents based on the collation of tests on concrete over a ten year period. The tests were designed to study plastic flow in concrete in more detail by varying both loading and concrete mixtures, including variations in load intensity and duration, concrete moisture content, aggregate characteristics and reinforcement, amongst others. The experimental work was subsequently refined by Troxell (1958) and was later published by the American Society for Testing Material as a significant database of creep and shrinkage data.

In 1943 Ross published a paper on the effects of shape, size and moisture-loss of mortar[5]. Ross made some of the first efforts to investigate the effect of size and shape on mortar specimens. Four main cross sections of varying size were investigated with the aim of testing the applicability of diffusion and surface emission equations. This was the first significant work performed on a form of concrete of varying shape and enabled Ross to begin to determine the relevance of the surface to volume ratio of a given section.

In 1958 at the 61st Annual Meeting of the American Society for Testing Materials, research from the University of California was presented relating to creep and shrinkage of both plain and reinforced concrete based on test data collected over a period of up to 30 years (Troxell et al 1958). The study was comprehensive, taking into account almost every possible variable associated with concrete and concrete reinforcing from size of specimens and their aggregates to curing conditions. The findings were therefore able to show the complexities of creep and shrinkage. Due to the longevity of the testing period the findings confirmed previous analysis of creep and shrinkage with no major contradictions.

[5] Mortar is a mixture of cement and water

Previous to the presenting of this study, Ross (1943) had published a paper on creep of concrete under variable stress. This was significant as most of the work on creep had considered constant stress. Ross developed three methods for computing creep under variable stress from available creep data obtained by considering constant stress. Firstly, Ross introduced the idea of an effective modulus, which simply allows the modulus of elasticity to be changed with time, based on the amount of specific creep[6] and thus enabling the calculation of stresses with time. A second method suggested is the *rate of creep* method where the rate of creep is calculated from the specific creep curve. The third suggested method is superposition. The principle of superposition for creep is an important one. Given that creep is proportional to stress within the ranges from 0% to 50% of ultimate stress, any variation of stress within the allowable range in either direction can be estimated by superimposing the correct creep curves (Ross 1958). Unfortunately concrete never shows a complete recovery, with a permanent deformation usually experienced after stress removal. McHenry (1943) restated the principle of superposition to make suitable allowance for reducing creep, and by doing so showed that concrete obeys the principle of superposition, albeit in a modified form.

Based on the historic database from Troxell et al (1958) a number of significant advancements were made in concrete research. Pauw (1960) published a paper on the static modulus of concrete where he derived an empirical formula for the modulus of elasticity for concrete as a function of 28-day strength and density. The derivation is all inclusive and is one of the first formulas to date that allows for lightweight concrete and is backed up by extensive test data. Only very weak concrete shows large discrepancies with test data, which as the author pointed out is not of great concern as such concrete is rarely used for structural columns.

Until the mid 1960s most of the creep and shrinkage testing of concrete samples was based on concrete cylinders (except for those by Ross who considered mortar rather than concrete). Although various sizes were used, the simple fact that all samples were circular and of similar proportions made it difficult to say with any conviction that creep and shrinkage is a function of size and/or shape. In 1966 Hansen and Mattock made a significant breakthrough when they tested samples of both circular and I-shaped

[6] Specific creep is a term referring to the creep per unit stress

sections that had up to a 600% variation in size. Through controlled testing over 4 years they could conclude that the volume-surface ratio is a real variable to consider in creep and shrinkage models. Hansen and Mattock (1966) also published a set of curves for both creep and shrinkage as functions of their maximum values against time for different volume to surface ratios.

In 1970 the ACI committee 209 developed a model for predicting creep and shrinkage of concrete, for which creep coefficients are obtained through the product of a number of independent coefficients based on certain concrete properties. In the same year the CEB/FIP[7] (1970) made recommendations on how to determine concrete creep and shrinkage. Their method also handled creep coefficients as a product of a number of independent coefficients. This model was later to change with a new method proposed in 1978.

Bazant (1968-1995) began work in the late 1960s on creep and shrinkage models. He is one of the leaders in the area of formulation of creep and shrinkage models for modern concrete. A paper on the age-adjusted modulus for concrete proposed a very important refinement of the static modulus of elasticity, devising an effective modulus method that accounts for concrete aging. He introduced and defined the so-called aging coefficient that is commonly used in current creep analysis. Whilst in normal structural design, Pauw's (1960) modulus of elasticity is usually sufficiently accurate, when considering creep and shrinkage effects the age-adjusted effective modulus is far more accurate as considerable degrees of shrinkage occurs while concrete cures. Pauw's modulus is only based upon samples of 28 days old.

In follow up work, Bazant and Panula (1972) concentrated on the development of practical models for predicting creep and shrinkage. Three series of papers, six parts in total, were published in which a model for prediction of creep and shrinkage was formulated. The first of the publications (parts 1 and 2) develops a practical model for predicting creep and shrinkage. The main variables involved in the model (BP, Bazant and Panula model, also known as BaP model) are concrete composition, strength, shape,

[7] CEB/FIP English translation for Comite Euro-International du Beton

size and age at loading. In the second publication, drying creep[8] and temperature effects are studied in detail and an extension to the practical model developed earlier is formulated. In the third publication Bazant looked further at temperature effects on drying creep and at cyclic creep, non-linearity and statistical scatter.

The CEB/FIB model (1978), the British Concrete Society BCS model (1978) and a German model DIN 4227 (1979) were all published around the same time as the BP model (1978). With the exception of the British model, which is the simplest, all of these models handle creep coefficients by summing individual strain components linked to different creep mechanisms. While this tends to complicate calculations without making them significantly more accurate, there is a greater potential for improvement with such a method (Muller et al 1982).

With a number of concrete creep and shrinkage models formulated and with more experimental data becoming available, there was good reason to make an objective comparison between the prediction methods and experimental data. In 1982 Muller and Hilsdorf compared six of the major methods (Muller et al 1982). Their tests only considered normal weight concrete of normal Portland cements for 50 samples which were tested for periods between one and three years. Long term creep values were extrapolated by hyperbolic creep-time expressions. In all cases, there were substantial discrepancies between predicted creep and experimental values. The authors concluded that there was room for improvement and suggested such in the optimization of the methods, and that *'upper and lower bounds really should be considered'* (Muller et al 1982).

Brooks (1984), using data collected over a much longer time frame, made a better comparison of five models for predicting creep and shrinkage. He took his comparisons further to include a 30 year prediction of strains. Data was also collected over a much longer period, thus the inaccuracies of extrapolation were removed. Brooks looked at 18 different concrete types, some lightweight and all with varying water-cement ratios. The samples were cured in both air and water. Swelling of concrete samples was also considered. He compared how all of these models predicted 30 year creep when using

[8] Drying creep is a term defined by Bazant as creep in a drying environment

short term values for creep and using the more accurate data collected over ten years. The difference was unsatisfactorily large. To calculate basic creep, shrinkage and total creep the BP model was used with inaccuracies of up to 35%. Even still this was one of the better performers.

Bazant, Kim and Panula (1991-95) published six papers on improved prediction models for concrete. The six papers were each a part of what was to be known as BP-KX model[9]. Each part revised one of the six parts from the previous BP model. The authors attributed the improvements to both the availability of further experimental data and an improvement in the knowledge of physical concepts and mechanisms. A number of new concepts as well as new coefficients for creep and shrinkage were developed.

Possibly one of the more significant advancements in concrete creep and shrinkage research began in Spain in 1993 when a symposium on creep and shrinkage proposed a data bank on concrete creep and shrinkage. Out of this process RILEM published a new model for creep and shrinkage known as the B3 model (Bazant 1995)[10]. The model is actually the third major update of the BP and BP-KX models. The changes to the BP-KX model do not seem to be significant although the model has been simplified and the authors state that it both '*agrees better with the experimental data and is justified better theoretically*' (Muller 1982).

To date, the B3 model is perhaps the best constitutive model available, a point well made by Koutsoukis (1995) when he concludes that '*the Trost-Bazant Age-Adjusted Effective Modulus Method combined with the 0.8 average value aging coefficient and the Bazant Baweja concrete property model is potentially the most sophisticated of methods presently available*'. This conclusion is reached after extensive probabilistic research on nine constitutive models compared to data collected for the Bayoke Tower II building in Thailand.

---

[9] Also known as BaP1 model

[10] The model is prepared by Bazant and Baweja

At present the main areas of research still lacking with respect to creep and shrinkage modelling is in the collection of data and the development of, or refinement of, existing models for creep at various stress levels, including relaxation and recovery.

1.6.2 Studies of major importance on TCBs

The first important paper concerning the application of creep and shrinkage theory was produced by the Reinforced Concrete Association (Seed 1945). The publication is titled *A Review of Recent Progress* and looks at the processes of creep and shrinkage and what can be expected by the designer. One of the features of this paper is its explanations of creep and shrinkage and the mechanics of each process. Other than this, the paper does not give any real insight into design for creep or shrinkage, rather it simply points out what to expect. The application of creep and shrinkage to tall structures was improved with the work of Fintel and Khan (1969- 1986).

Fintel and Khan (1969) published a paper that was directly related to the effects of column creep and shrinkage in tall structures. They developed a procedure for predicting creep and shrinkage in columns, considering load history, volume to surface ratios and reinforcement effects. They also tried to model the loading history of columns in a multi-storey structure, taking into account the phases of construction. Although there is no mention of studies on actual buildings in the paper, the authors made a few interesting points about the mechanics of creep and shrinkage in multi-storey buildings. The most interesting are;

i) that the final inelastic strains in reinforced columns and walls have much less variation due to the restraining effect of the reinforcement
ii) that elements with substantial loading at early ages, such as columns in the upper stories, are prone to higher shrinkage and creep than lower storey columns

For this the authors attribute incremental loading, larger sections and greater reinforcing as the reason.

Fintel and Khan (1971) published a more comprehensive paper on differential shortening of columns from a six year observation of a 38 storey building. In this paper an attempt is made to predict the framing action of walls and columns based on what the authors call an *'equivalent one bay frame'*, where charts are used to estimate differential shortening. The method suggested for calculating differential shortening is quite basic and essentially involves summing all the axial shortening values of each element for each level, based on the time since construction.

Warner (1975) published a report on axial shortening in reinforced concrete columns. He proposed two methods for calculating column shortening, one a simpler hand method and a more detailed method intended for computer calculation. The data has been presented in such a way that it could be used in conjunction with creep data from AS1481 (1974). The procedure is developed using the fundamental theorem of superposition which allows Warner to use an incremental method of analysis. The total time in question is broken up into a number of intervals coinciding with the external load changes. The significance of these intervals is that changes in external loading must coincide with these intervals.

Warner's view is that *while improvements can be made, it is arguably the most up-to-date on the subject of axial shortening with framing action'*. Here he considers the effect of framing for both elastic members and viscoelastic[11] members. The significance of this paper is not in the actual calculations made but in the model that Warner uses. Warner's model is applicable to TCBs, stiffened by lift shaft cores or equivalent massive elements. For such cases external supports prevent rotation of the beam-column joints. These provide moment reactions for the beams and hence prevent significant bending moments from entering the columns.

In 1984 one of the first major case studies on axial shortening of TCBs in Australia was published. The study was conducted at the New South Wales Institute of Technology on one of the new buildings build on campus. The results published were for a period of

[11] Viscoelastic refers to framing members that don t follow full elastic behaviour. In the case of concrete at sufficient bending moment levels the section cracks and the behaviour is no longer elastic.

investigation of over ten years. The measured strains were compared to theoretical predictions made by Fintel and Khan. Significantly the theoretical predictions over estimated shortening by 10% to 30%. As little or no attempt was made to account for actual framing effects the authors believe that the over prediction of actual axial shortening by the methods proposed by Fintel and Khan can be attributed in part to the inaccuracies associated with the method adopted to take into account framing action. This study highlights the need for better models to predict axial shortening of framed structures.

Further substantial work on axial shortening of concrete columns was published by Beasley (1987) as a research report that became the basis of a program to calculate differential axial displacements between cores and adjacent column. The report develops equations for the calculation of creep, shrinkage, elastic and relaxation components of axial deformation. These are used in conjunction with an idealised building cycle to find differential displacements at each level of a multi-storey building. The results of three popular standards on concrete creep and shrinkage properties are compared.

In 1994 Koutsoukis and Beasley published a paper on axial shortening involving a Monte Carlo analysis of deformations. The aim of the paper was to determine the overall significance of a number of random variables associated with axial deformations. The analysis was conducted on the BA-XF model. The paper makes an important conclusion, amongst others, *'that the variation in differential shortening results is greatly effected by basic shrinkage strain variation* ' (Koutsoukis 1994).

The same authors extended the statistical analysis to eight models. Aging coefficients, and constitutive concrete models are all compared by statistical methods fitted to Monte Carlo simulation. The results of a 39 storey building are used to compare the models under investigation. The authors could also determine the more sensitive variables associated with each model as well as how closely they predicted actual behaviour.

Work by Koutsoukis and Beasley (1997) is collected together and appraised in one research paper. One major addition to previous studies is a statistical analysis of the B3

model by Bazant (1995). Koutsoukis's findings appear to confirm Bazant's assertions that the B3 model is the best model currently available for predicting column shortening behaviour.

Bursle (2003) has begun research on the effect of high strength concrete and is attempting to determine the effect of concrete strength on axial shortening by taking measurements from the World Tower in Sydney. The work considers actual average compressive strength against the predicted compressive strength specified and how this affects theoretical predicted axial shortening. In many cases the predicted strength specified using ACI codes is considerably less than the actual measured strength.

Currently the development of models to determine axial shortening in multi-storey structures revolve around axial deformations of isolated concrete elements. Although attempts have been made to account for framing action, at present there is no definitive approach that aims to evaluate differential axial deformations for framed concrete structures.

Chapter 2

CREEP AND SHRINKAGE

2.1 Introduction

When designing multi-storey structures from reinforced concrete it is essential that the designer is aware of how the material behaves. In the case of axial shortening, long term effects like creep and shrinkage are important characteristics of concrete that require modelling so analytical results can be made. Incorporating these models into full scale analysis of a multi-storey structure requires a defined approach. In order to broaden the applicability of the model, and allow us to identify exact discrete points in time where the structure experiences a fundamental change during the construction process, a construction cycle for a multi-storey structure is required. With such a cycle we can approximate the expected time that each element is constructed and at what stage the element experiences additional loading when any framing action is introduced. Section 2.2 details the relevant theory on concrete properties and how these properties contribute to axial shortening in multi-storey buildings.

2.2 Relevant Concrete Properties

2.2.1 Curing

The process of curing is fundamental to the mechanics of creep and shrinkage. Curing of concrete begins at the first stages of creation. As soon as water comes in contact with cement in a concrete mix it causes the concrete to begin to hydrate. Initially heat causes the dissolved cement particles to transform into calcium silicate gel (Faber 1927). This gel acts as a bonding agent between the aggregates in the concrete mix. In order to make a concrete mixture workable there is usually more water present in the mixture than the necessary amount required to form the bonding gel. Although some of the extra water is

often lost through the boundary walls of the concrete mould or is evaporated from the surface, a large quantity remains in the concrete mix. With time and heat the calcium silicate gel undergoes a chemical change to form calcium hydroxide crystals (Faber 1927). This traps gel and loose water inside concrete pores. Both loose water and water from the gel are lost over time. The speed of this transfer is one of the major factors that influence creep and shrinkage (Faber 1927).

2.2.2 Shrinkage

The properties of the cement gel form an integral part of the phenomenon of concrete shrinkage. As a binding agent the cement gel when wetted will take up liquid and then swell, whilst when drying the gel will give up liquid and contract (Seed 1948). The properties of mortar[1] allow it to shrink and swell in-elastically[2] for a number of years after creation. Therefore if we assume that most aggregates added to this mortar do not deform in-elastically, we can attribute the measurable volume changes in concrete shapes to movements in the mortar (Ross 1943).

Research on concrete samples reveals a moisture gradient across a concrete section as it cures (Neville 1983). This is attributed to surface gel losing water due to evaporation and replacing this with free water from within the concrete pores (Pickett 1946). Thus the majority of shrinkage occurs near to the surface, while closer to the centre of a concrete member the shrinkage is much less (Hansen et al 1966). Figure 2.1 shows how volume to surface ratios effect shrinkage[3]. The curves show variations for different volume/surface ratios for 50% humidity and are expressed as a ratio of shrinkage strain to ultimate shrinkage strain.

---

[1] Mortar as defined by Ross (1943) is the cement paste formed from water and cement dust

[2] In-elastic deformation is a deformation that is not linearly related to stress applied, thus we would expect no deformations in aggregates if no load is applied

[3] The same variation is equally applicable to creep

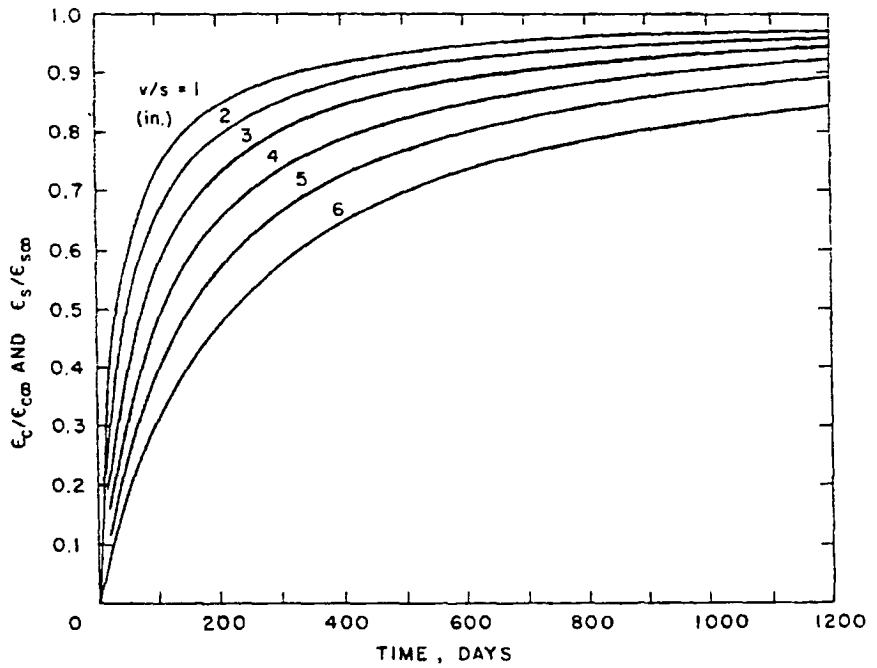


Figure 2.1 Calculated variation of shrinkage and creep with time for given volume/surface ratios as published by Hansen and Mattock (1966)

where (as defined by Hansen and Mattock (1966));

$\varepsilon_c/\varepsilon_{c\infty}$ = creep strain to maximum creep strain ratio

$\varepsilon_s/\varepsilon_{s\infty}$ = shrinkage strain to maximum shrinkage strain ratio

v/s = volume to surface ratio

Shrinkage strains have been shown to be a function of four components (Neville 1983). The mean total shrinkage strain (ε_{sh}) is the only value of importance when making axial shortening calculations. Further discussion on shrinkage strain can be found in Appendix A2.1

2.2.3 Creep

Like shrinkage, creep effects in concrete are due to how water is stored within concrete. It has been shown that plastic flow contributes to about 10% of creep while the rest is affected by the moisture conditions and the age of the concrete (Ross 1943). The majority

of creep can therefore be attributed to the cement gel and how it changes. Experiments have shown that it is possible to squeeze water from cement gel by applying pressure (Ross 1943). The process of squeezing water from a concrete sample can be compared to seepage, where water under pressure moves away from high areas of pressure, with a result that stresses are distributed to the solids (Ross 1943). Thus as water moves from high areas of pressure, the tendency is for these areas to reduce in volume and result in creep of the sample. Experiments by Davis (1937) confirm this phenomenon.

It seems quite clear that the factors that affect shrinkage will also affect creep as the majority is due to water movements within concrete with the only addition being that creep is also a function of stress. Further discussion on creep strain can be found in Appendix A2.2.

2.2.4 Superposition

The principle of superposition is fundamental to the process of incremental analysis of creep and shrinkage. It is a well known principle that applies to the analysis of structures constructed from non-aging[4] materials. However, when taking a long term look at concrete structures which experience changes in properties as they age, the use of the principle of superposition cannot be assumed and needs further evaluation. The principle of superposition allows us to add together increments of strain as they are experienced at various stages, enabling evaluation of the total strain over the life of a structure. When an applied load is monotonically increased, experimental data for aging materials, such as concrete, complies with the principle of superposition (Troxell et al 1958).

McHenry (1943) was the first to publish material on creep and recovery for concrete. He postulated an expression for the principle of superposition as:

> *'The strains produced in concrete at any time (t) by a stress increment applied at any time (t_0) are independent of the effects of any stress applied either earlier or*

[4] Here non-aging refers to materials that do not change their properties with time

later than at (t_0). The stress increment may be either positive or negative, but the stresses which approach the ultimate strength are excluded'.

This premise was tested by applying stress via an axial load to a concrete section at 28 days and then removing the load at 90 days. The results showed a reversal in creep of the same magnitude, but opposite to that measured for another concrete section with the same stress applied from day 90 onwards. Refer Figure 2.2. The experiment is fundamental when considering that framing effects in multi-storey structures can cause similar load reversal.

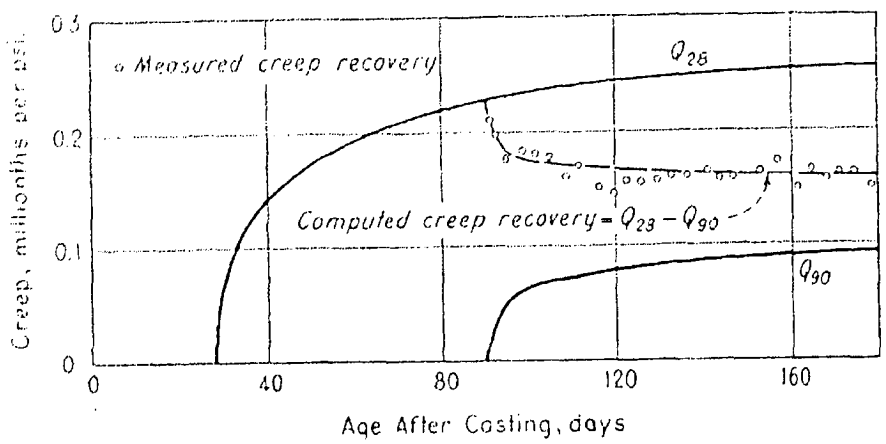


Figure 2.2 Creep and creep recovery in a concrete cylinder from plotted data by McHenry (1943)

Further experimental work by Bazant (1988) enabled limits to be imposed to the validity of creep and shrinkage models due to the conditions of the application of the principle of superposition. If the limits imposed by Bazant are satisfied, the calculation of total axial strain for an isolated member can be found from the sum of each individual strain applied independently. Load accumulation and reversal can therefore be added based on the principle of superposition for concrete.

2.3 Models

2.3.1 Creep and shrinkage coefficients

The modelling of creep and shrinkage is essential when we consider the axial shortening of TCBs. To date there are numerous models available for calculating creep and shrinkage coefficients. Koutsoukis (1997) has investigated ten of the most advanced models and, by a rigorous statistical analysis of each model, concludes that the B3 concrete and shrinkage model (Bazant 1943) is potentially the most sophisticated of all approaches at present. AS3600 (2001) contains models for creep and shrinkage coefficients. Gilbert (2002) has published research on creep and shrinkage models for high strength concrete as a proposal for inclusion in AS3600 (2001) to expand the scope of the code to cover concrete with characteristic compressive strengths up to 100 MPa.

Another model for determining creep and shrinkage coefficients is the model developed by the American Concrete Institute (ACI 1986). A reliable computer program COLECS (1987) exists for predicting isolated axial deformations using this method. To allow both programs to be correlated the development of a working program to calculate axial shortening involving framing action is made using the ACI (1986) method for determining creep and shrinkage coefficients. Although incorporating the creep and shrinkage coefficients as modelled by Gilbert (2002) into the program is not arduous, the main emphasis of this thesis is to provide a solution to framing action rather than to compare creep and shrinkage models. The development of the model used in this research to calculate axial shortening for framed structures based on ACI (1986) is given below.

The two fundamental equations for calculating shrinkage and creep and by ACI (1986) are given by the two expressions respectively

$$\varepsilon_{sh}(t) = \frac{t_d}{35 + t_d}\,\varepsilon_{sh}(u) \qquad (2.1)$$

where

$\varepsilon_{sh}(t)$ = time related shrinkage strain
$\varepsilon_{sh}(u)$ = ultimate shrinkage (micro-strain)
t_d = age of concrete from onset of drying

$$\phi_t = \frac{(t-s)^{0.6}}{10+(t-s)^{0.6}}\ \phi_u \qquad (2.2)$$

where

ϕ_t = time related creep coefficient
ϕ_u = ultimate creep
t = total age of concrete
s = age at loading

$\varepsilon_{sh}(u)$ and ϕ_u are both made up of a number of multiplying factors. These factors are determined by the following characteristics of a concrete element

i) volume to surface ratio
ii) relative humidity
iii) concrete slump
iv) percentage of fine aggregates
v) percentage of air
vi) initial curing period
vii) cement content

The basic relationships between dependent and independent parameters are given in Appendix A2.3.

2.3.2 Age-adjusted effective modulus

Regardless of how creep and shrinkage coefficients are obtained, the most current models for calculating total axial strains are based on the age-adjusted effective modulus (Bazant 1972). The development of an equation to calculate axial strain is based on the constitutive equation for concrete. A full derivation of the age-adjusted effective modulus and the equations to calculate axial strain is given in Appendix A2.4.

The constitutive equation for strain can be expressed as

$$\varepsilon(t) - \varepsilon_{sh}(t) = \alpha + \beta\,\phi(t,s) \quad \text{for } t > s > 0 \qquad \text{(Bazant 1972)}^5 \tag{2.3}$$

where

$\varepsilon(t)$ = total strain at (t) days from casting of concrete

$\varepsilon_{sh}(t)$ = total shrinkage strain at (t) days from casting of concrete

α , β = are constants

ϕ (t,s) = creep coefficient, where (s) is the time at load application and $t > s$

The effective modulus for concrete, defined by Ross (1958), is given as

$$E'(t,s) = \frac{E_c(s)}{1 + \phi\,(t,s)} \tag{2.4}$$

where

$E_c(s)$ = modulus for the concrete section at the application of load

All other symbols have the same meaning as defined previously

The age-adjusted effective modulus, defined by Bazant (1972), is given as

$$E''(t,s) = \frac{E_c(s)}{1 + \chi(t,s)\,\phi\,(t,s)} \tag{2.5}$$

where

$\chi(t,s)$ = aging coefficient

All other symbols have the same meaning as defined previously

By re-arranging the constitutive equations for strain and the definitions of concrete modulus by Ross (1958) and Bazant (1972), the equation for determining strain in a concrete element at time (t) is

$$\varepsilon(t) = \frac{\sigma_c(s)}{E'(t,s)} + \varepsilon_{sh}(t) + \frac{\Delta\sigma(t)}{E''(t,s)} \tag{2.6}$$

Equation 2.6 forms the basis of the majority of models used to determine axial deformations in concrete and is fundamental to the development of the models adopted in this project.

---

[5] Some symbols have been altered to reflect the nomenclature used in this report

2.3.3 Reinforced concrete model

In the case of multi-storey structures it is only the axial loading effects on a *reinforced-concrete* column that are of interest. Therefore an analytical model for a *reinforced-concrete* element needs to be determined. Similar models to that derived below have been developed by Beasley (1987), Gilbert (1998-2002), Koutsoukis (1995) and Samra (1989).

The full derivation of the equations required to predict axial shortening values for a reinforced concrete column are given in Appendix A2.5. Due to the applicability of the resulting equations for framed structures, the derivation is similar to that of Koutsoukis (1995).

In this derivation a number of assumptions have been made;

i) the axial loading is assumed to be uniform throughout the length of the column
ii) the axial loading is assumed to be uniform across the full section of the column
iii) the reinforcement is of sufficient length and bond over the entire length of the column that full development of the reinforcement can be achieved without slip

The axial shortening experienced by multi-storey structures contains both short and long term components. The four components of axial shortening, that are referred to as discrete axial deformations in this research can be expressed as;

i) $\delta_e(s)$ elastic
ii) $\delta_c(t,s)$ creep
iii) $\delta_s(t)$ shrinkage
iv) $\delta_r(t)$ reinforcing stiffening component

To develop the model here, consider the following

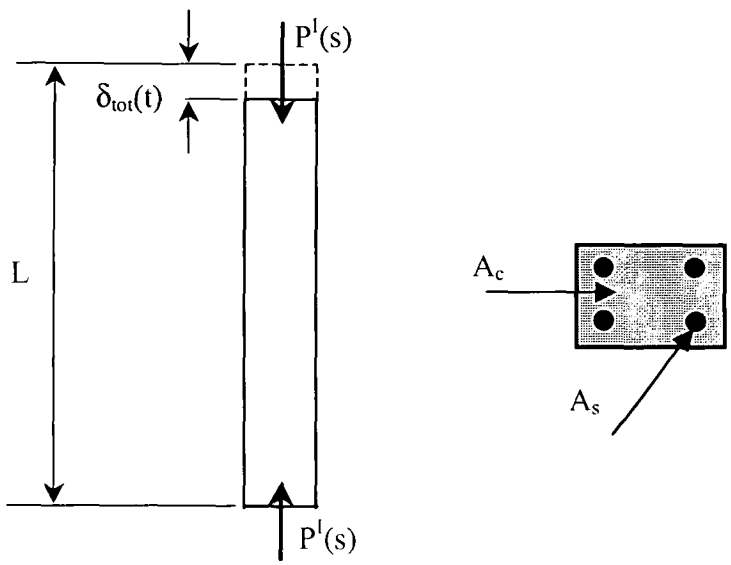


Figure 2.3 Axially loaded reinforced concrete column

where

$P^{I}(s)$ = initial applied load at time (s), assumed uniform over the section
A_c = area of concrete
A_s = area of steel
$\delta_{tot}(t)$ = total time dependant axial shortening of the column at time (t)
L = total effective length of the column with constant axial load P^{I}

Based on (2.6) and the assumptions above, using the nomenclature introduced in Figure 2.3, the equation for the strain in a reinforced concrete section is;

$$\varepsilon(t) = \frac{P^{I}(s)}{A_c(1+n(s)p)\,E'(t,s)} + \varepsilon_{sh}(t) + \frac{1}{(1+n''(t,s)p)\,E''(t,s)}\left[\frac{P^{I}(s)}{A_c} - \frac{P^{I}(s)(1+n'(t,s)\,p)}{A_c(1+n(s)\,p)} - \varepsilon_{sh}(t)E_s p\right] \quad (2.7)$$

Refer Appendix A2.5 for full derivation.

By the definition of axial shortening $\delta_{tot}(t) = \varepsilon(t) \cdot L$ and rearranging (2.7)

$$\delta(t) = \underbrace{\frac{P'(s)L}{A_c(1+n(s)p)}\left[\frac{1}{E'(t,s)} + \frac{p\,(n(s)-n'(t,s))}{(1+n''(t,s)p)\,E''(t,s)}\right]}_{A(t,s)} + \underbrace{\varepsilon_{sh}(t)L}_{B(t)} - \underbrace{\frac{\varepsilon_{sh}(t)E_s pL}{(1+n''(t,s)p)\,E''(t,s)}}_{C(t)} \quad (2.8)$$

where

A(t,s) represents the discrete elastic and creep component of axial shortening ($\delta_e(s) + \delta_c(t,s)$)

B(t) represents the discrete shrinkage component of axial shortening ($\delta_s(t)$)

C(t) represents the associated discrete reinforcing stiffening effect ($\delta_r(t)$)

At the application of the load (i.e. time (s)) the creep component associated with the applied load $P'(s)$ is 0, thus from A(s,s)

$$\delta_c(s) = \frac{P'(s)L}{A_c(1+n(s)p)}\left[\frac{1}{E'(s,s)} + \frac{p\,(n(s)-n'(s,s))}{(1+n''(s,s)p)\,E''(s,s)}\right] \quad (2.9)$$

and

$$E'(s,s) = E''(s,s) = E_c(s)$$

thus

$$n(s) - n'(s,s) = 0$$

therefore

$$\delta_e(s) = \frac{P'(s)\,L}{A_c(1+n(s)\,p)E_c(s)} \quad (2.10)$$

and

$$\delta_c(t,s) = \frac{P'(s)L}{A_c(1+n(s)\,p)}\left[\frac{1}{E'(t,s)} + \frac{p\,(n(s)-n'(t,s))}{(1+n''(t,s)\,p)\,E''(t,s)}\right] - \frac{P'(s)\,L}{A_c(1+n(s)\,p)\,E_c(s)} \quad (2.11)$$

additionally

$$\delta_s(t) = \varepsilon_{sh}(t).L \tag{2.12}$$

At the onset of drying the stiffening due to reinforcement begins, thus

$$\delta_r(t) = \frac{\varepsilon_{sh}(t)\, E_s\, p\, L}{(1+n''(t,t_0)\, p)\, E''(t,t_0)} \tag{2.13}$$

where (t_0) is the age in days at onset of drying.

2.4 Application of Models to TCBs

As creep and shrinkage both begin from the first day of construction it is important that the building cycle, which may take years to complete, is investigated to allow a procedural model to be formulated and to be implemented when making axial shortening calculations.

The vast majority of multi-storey buildings are built around a core. The core contains the service ducts and the lift shafts and is of substantial size and rigidity due to lateral wind load resistance as well as gravity load resistance. The core will usually be constructed first and may proceed the rest of the structure by some number of storeys. From the core, the floors and the supporting structure are constructed (Beasley 1987).

In the development of COLECS, Beasley (1987) models a typical building over the initial 12 stages. Refer Figure 2.4 below. Defining these stages allows the building process to be broken down into discrete parts. The 12 stages show the construction of floors but do not mention framing connections. In some cases the connections between the core and the remaining structure will be rigid. Details of connections are discussed in Chapter 3.2.1.

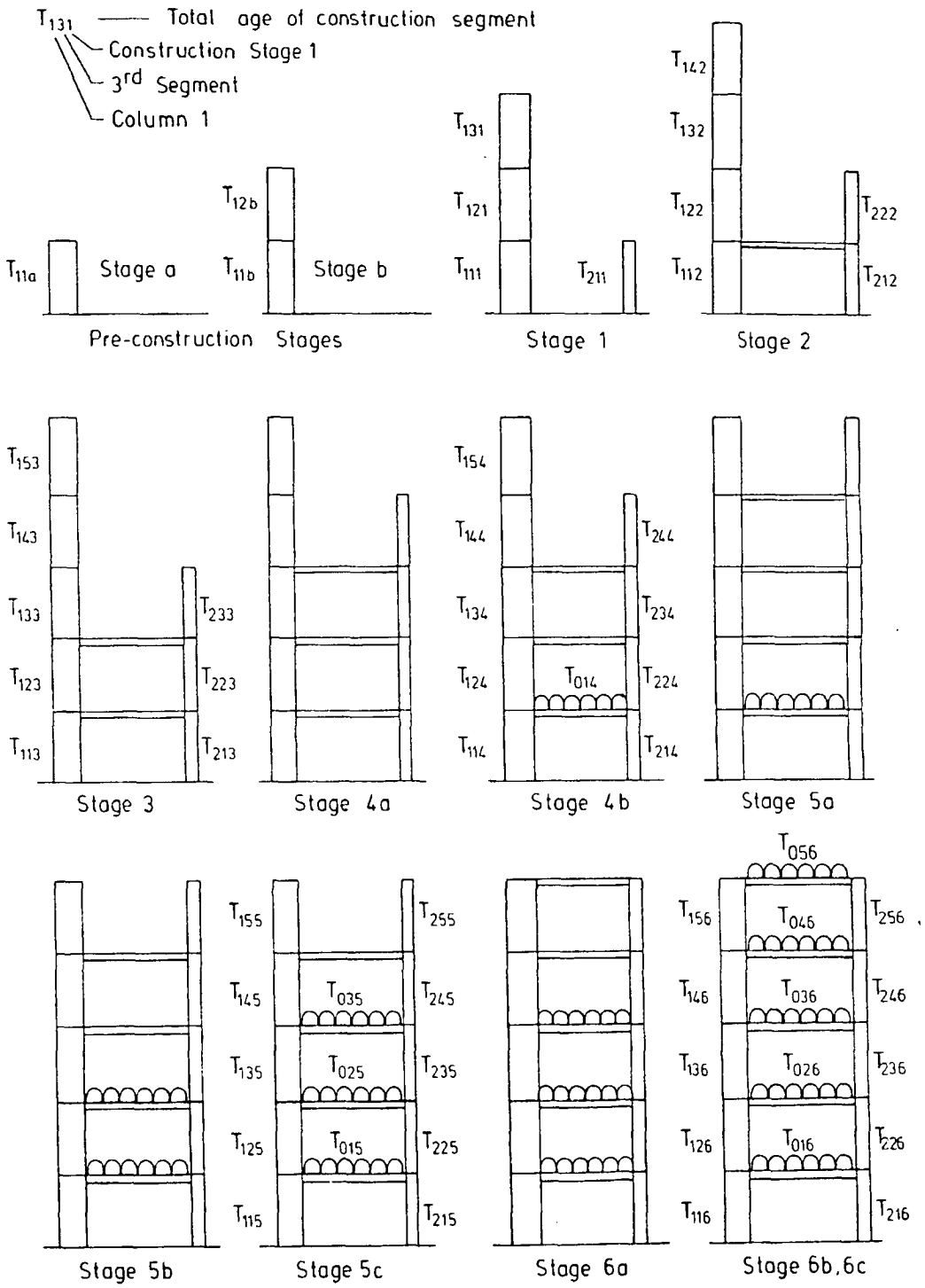


Figure 2.4 The staged construction of a nominal building as proposed by Beasley (1987)

When making an axial shortening analysis for a framed structure, a model of the building cycle must be able to determine all the points in time when a section of the core is added, a new column or secondary element is added and at what point in time are any of these elements loaded. Additionally it must be able to identify the point in time when framing action occurs, and between which two elements. This relationship is essential as from that point the two elements no longer continue to act independently. Whilst in some cases the addition of a framing element may occur at the same time a new loading is added to the structure, at other times it does not.

The introduction of the framing element fundamentally changes the creep and elastic axial shortening components of the total shortening of any element. All of these points represent fundamental changes to the structure as the total axial shortening increases at any one of these points.

Most buildings will generally follow a basic construction cycle similar to that indicated in Figure 2.4, where one can identify regular periods in time whence a certain amount of work on the structure is carried out. Although the use of a construction cycle allows for faster and easier calculation of axial shortening, it does not cover the events where a change in the cycle, or a different cycle, is introduced. It also does not allow certain important events to be pinpointed during the construction phase.

A more general approach that still recognizes the building cycle of Figure 2.4 is needed to accurately define the point in time when framing action can be measured.

If we consider a number of applied loadings at discrete intervals s_1, s_2... ...s_n then by applying the principle of superposition (McHenry 1943, Bazant 1988)

$$\delta_{tot}(s_n) = \sum_{i=1}^{n} \delta(s_i) \qquad (2.14)$$

Consider Figure 2.5 below

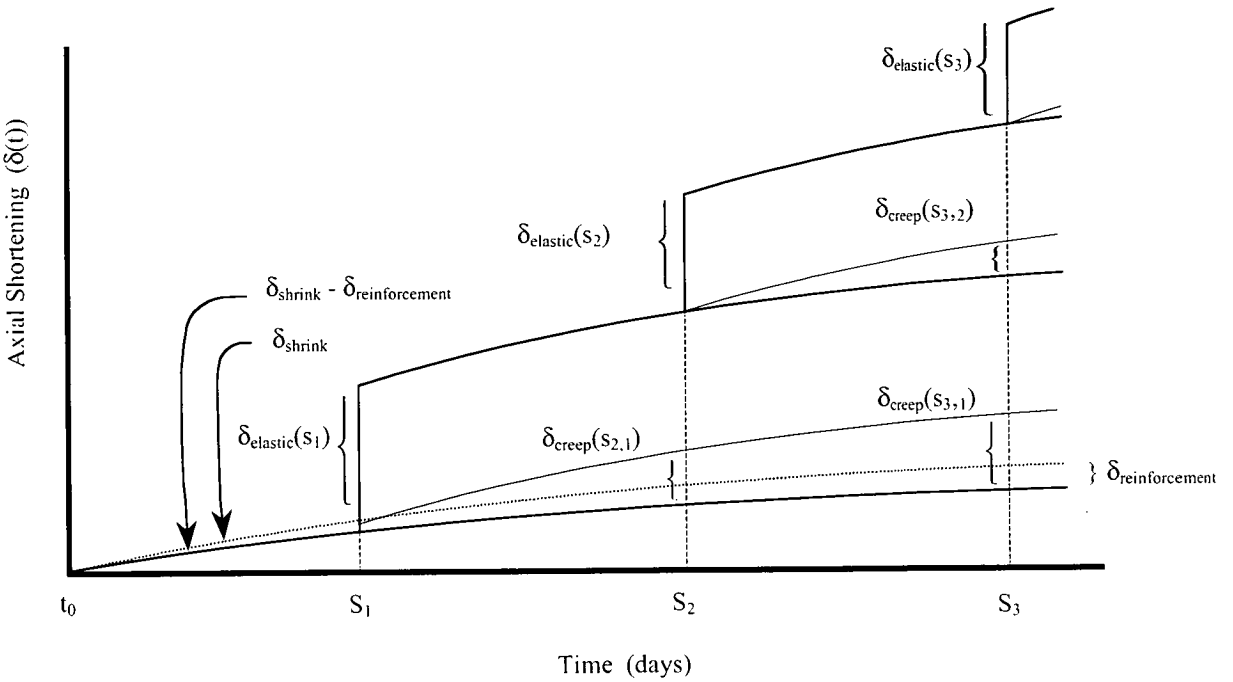


Figure 2.5 Axial shortening values of a reinforced concrete column with applied loads at time (s_1), (s_2), (s_3)

Therefore from Figure 2.5 and (2.14) at (s_3) the total axial shortening is

$$\delta_{tot}(s_3) = \underbrace{\delta_e(s_1) + \delta_c(s_{3,1})}_{A(s_{3,1})} + \underbrace{\delta_e(s_2) + \delta_c(s_{3,2})}_{A(s_{3,2})} + \underbrace{\delta_e(s_3)}_{A(s_{3,3})} + \underbrace{\delta_s(s_3)}_{B(s_3)} - \underbrace{\delta_r(s_3)}_{C(s_3)} \qquad (2.15)$$

All other discrete time intervals can be evaluated similarly.

Figure 2.5 represents the time related axial deformation of a single isolated element experiencing loading increments at time (s_1), (s_2), (s_3). In the case of an element linked by a rigid connection to one or more elements by a frame, the process becomes more complicated.

Creep and axial deformations are altered by the framing process. On account of this the creep and axial components of each discrete section, $A(s_{3,1})$, $A(s_{3,2})$, $A(s_{3,3})$, $B(s_3)$ and $C(s_3)$ from time (t_o) to (s_3) must be calculated separately.

If a rigid connection is introduced instead of the axial load at (s_s) then load re-distribution takes place due to the load sharing through framing action. Figure 2.6 demonstrates how a rigid connection will redistribute shear loading. The figure is for the arbitrary j^{th} storey. The structural mechanics of rigid connections and the development of the nomenclature are found in full in Chapter 3.

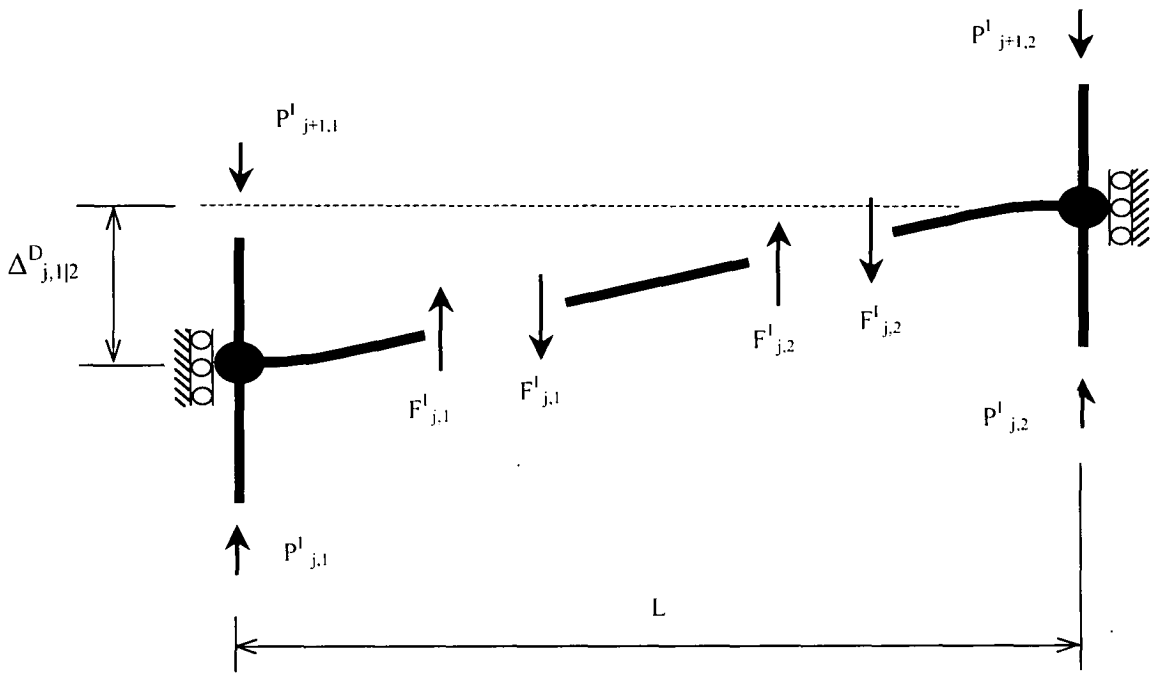


Figure 2.6 Basic structural mechanics of two rigid connections

where

F^I is the internal shear force at the vertical support due to the internal moment associated with the joint displacement

P^I is the internal axial force at the vertical support

L distance between the two joints

$\Delta^D_{j,1|2}$ differential axial shortening value of the column pair 1 and 2 at the j^{th} storey

by structural mechanics $$F^I_j = \frac{12 E_{1|2} I_{1|2}}{L^3} \times \Delta_{Dj,1|2} \qquad (2.16)$$

where

$E_{1|2}$, $I_{1|2}$ are the modulus of elasticity and the moment of inertial of the beam respectively

by equilibrium of forces at each node

$$P^{I}_{j,1} = P^{I}_{j+1,1} - F^{I}_{j,1} \quad (2.17)$$

$$P^{I}_{j,2} = P^{I}_{j+1,2} + F^{I}_{j,2} \quad (2.18)$$

$P^{I}_{j,1}$ and $P^{I}_{j,2}$ are the internal axial forces that affect axial shortening. At column 1 the axial load is reduced due to the transfer of shear forces, while at column 2 the axial load increases.

Figure 2.7 shows the same load history as Figure 2.5 but instead of an applied load at (s_2), the rigid connection shown in Figure 2.6 is introduced. The graph shows how creep shortening is affected by the rigid connection.

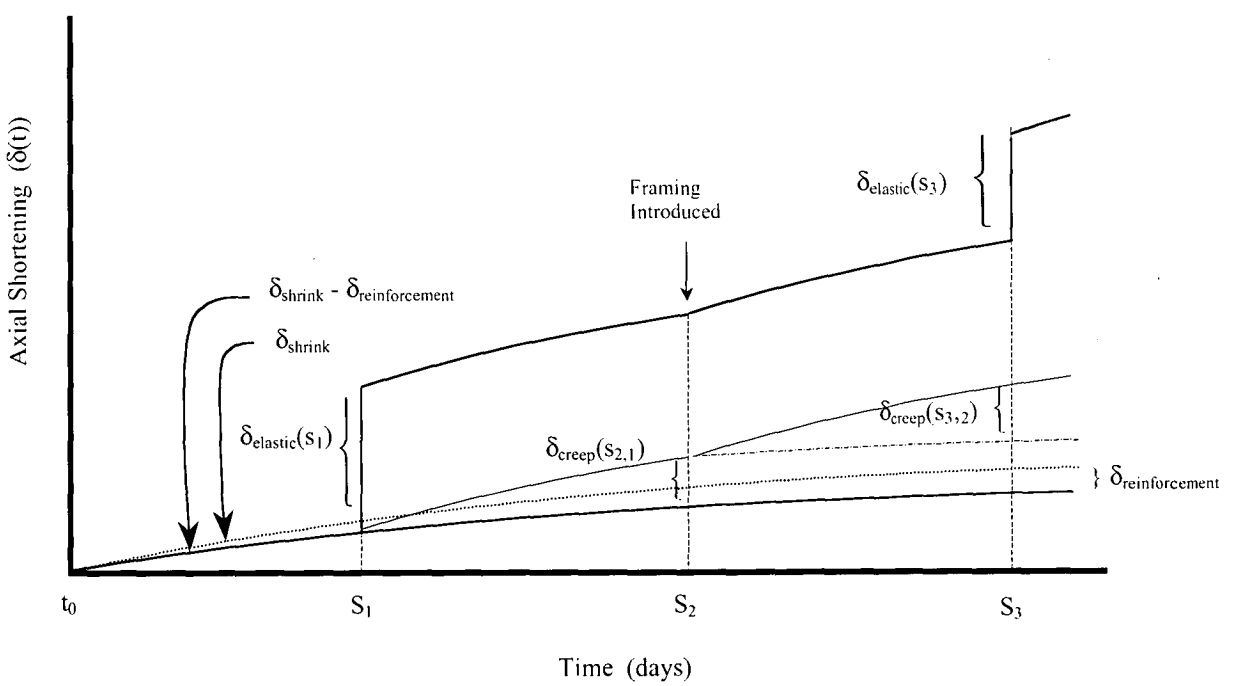


Figure 2.7 Axial shortening values of a reinforced concrete column with applied loads at time (s_1), (s_3) and framing effects introduced at (s_2)

Figure 2.7 shows how the principle of superposition must be applied when calculating the discrete sections of axial shortening. $A(s_{3,1})$ is now made up of two creep components. Unlike (2.15), where each must be calculated separately in order to obtain $A(s_{3,1})$.

Thus at (s_3) the total axial shortening is

$$\delta_{tot}(s_3) = \underbrace{\delta_e(s_1) + \delta_c(s_{2,1}) + \delta_c(s_{3,2})}_{A(s_{3,1})} + \underbrace{\delta_e(s_3)}_{A(s_{3,3})} + \underbrace{\delta_s(s_3)}_{B(s_3)} - \underbrace{\delta_r(s_3)}_{C(s_3)} \quad (2.19)$$

The major difference in Figure 2.5 is that without framing one can calculate the total creep shortening associated with the loading at (s_1) as one discrete value at (s_3). In Figure 2.7 however, the total creep shortening associated with loading at (s_1) must be calculated from the sum of two parts as the introduction of framing at (s_2) causes a significant change to the creep curve. Additionally, although not shown, all axial loading (i.e. at s_3) is affected by the framing introduced at time (s_2). The application of this figure and the development of the model used to determine framing effects is based on this important principle.

2.5 Summary to Chapter

The fundamental aspects relating to the relevant theory for the calculation of axial shortening values implemented in this thesis are presented. Of particular importance is the principle of superposition and how it is used to arrive at axial shortening values for framed members. A statistically proven model for determining long term creep and shrinkage in a concrete sample is derived in such a way as to be relevant to the analysis of axial shortening in multi-storey structures. The logical arrangement of each component of axial shortening of framed structures is provided to demonstrate where rigid connections cause a significant change to the mechanics of axial shortening. Analysing the components of the mechanics of axial shortening at all the discrete stages requires a structured approach. Each discrete stage is generic to one of the possible combinations of loading, alterations to the building, and the introduction of framing. It is imperative that each possible combination can be identified and accounted for in the correct manner at the correct point in time. The combinations affecting axial shortening of a framed TCB have been demonstrated in this chapter and all the axial shortening effects isolated. By isolating these alterations, further numerical analysis can now be made.

Appendix

A2.1 Shrinkage

Experiments have shown that shrinkage is less dependant on the amount of water that leaves a specimen than on the manner in which it is held (Seed 1948). This is demonstrated easily by considering a sample in reverse, that is, absorbing water and expanding. The rate of expansion does not depend on how much water is absorbed. In fact a large percentage of water is absorbed before any volume change can be measured, then even after saturation the sample continues to swell. It can be deduced that free water plays little part in the swelling and shrinkage of concrete, rather the change of moisture content of the gel and the creation of crystals determines the degrees of shrinkage (Seed 1948).

The types of shrinkage a concrete sample experiences over time consists of four components

$$\varepsilon_{sh} = \varepsilon_{ds} + \varepsilon_{cs} + \varepsilon_{hs} + \varepsilon_{cas} \qquad \text{(A2.1)}$$

where

ε_{sh} = total shrinkage strain

ε_{ds} = drying shrinkage

ε_{cs} = carbonation shrinkage

ε_{hs} = hydration shrinkage

ε_{cas} = capillary

As defined by Neville (1983), drying shrinkage is the basic moisture loss to the environment. Carbonation shrinkage, while usually small, is due to surface carbon dioxide at low humidity. Hydration shrinkage, which is usually the largest component, is when water is taken by the gel and crystals of $Ca(OH)_2$ form. Capillary shrinkage is used to describe the shrinkage that occurs when the concrete is plastic. With very wet mixes this can be quite high. Axial shrinkage models are only interested in total shrinkage (ε_{sh}). The total shrinkage will vary across a concrete section (Neville 1983) so the average value is adopted for engineering purposes when analysing axial shortening.

A large number of external conditions will contribute to shrinkage Neville (1983). However the six major factors, discussed in more detail by Neville (1983), Smerda (1989), and Bazant (1988) are;

i) humidity
ii) temperature
iii) cement type
iv) curing period
v) size and shape
vi) water cement ratio

When developing a model for shrinkage it is often necessary to only consider a fraction of these factors as efficiency often needs to be sacrificed for accuracy when trying to develop a mode for a real world situation. Appendix 2.3 deals with the development of the ACI (1986) creep and shrinkage model.

A 2.2 Creep

Like shrinkage, creep can be broken into components. The way it is broken down is dependent on how the process is defined. Basic creep and drying creep are two components of creep as defined by Bazant (1978). Here Bazant models creep in these two separate parts. Another method is to divide creep into recoverable and non-recoverable parts (Rusch 1983).

With the first approach total creep $\varepsilon_c(t)$ is simply

$$\varepsilon_c(t) = \varepsilon_{bc}(t) + \varepsilon_{dc}(t) \qquad \text{(A2.2)}$$

where

$\varepsilon_c(t)$ = total creep at time (t)
$\varepsilon_{bc}(t)$ = basic creep at time (t)
$\varepsilon_{dc}(t)$ = drying creep at time (t)

A modification of this, as adopted by the ACI (1986), expresses total creep in the form of

$$\varepsilon_c(t) = k_i\, \varepsilon_{bc}(t) \qquad \text{(A2.3)}$$

where

k_i = modification factor.

In order to determine both creep components, basic creep is found from samples that are sealed to reduce moisture loss, while drying creep in then the amount of measured creep over and above basic creep due to drying conditions.

In the case of dividing creep into recoverable and un-recoverable parts, elastic and flow components are considered, which are found experimentally (Rusch 1983). In this form the flow component in broken up into initial, basic and drying components. Basic and drying components are the same as for the first case, while initial creep is that which occurs within the first day after loading and is usually unrecoverable. The delayed elastic component is considered to be the portion of creep that can be recovered after loading is removed.

Both approaches are valid and have their distinct advantages and disadvantages.

A2.3 ACI Committee 209

Creep and shrinkage are both effected by surrounding environmental factors which include;

i) v/s volume to surface ratio
ii) λ relative humidity
iii) ξ concrete slump
iv) ψ percentage of fine aggregates
v) α percentage of air
vi) τ initial curing period
vii) ω cement content

These factors are handled by the ACI(1986) model by the use of a adjustment factor (k)

Basic time dependant shrinkage strain is defined as

$$\varepsilon_{sh}(t) = \frac{t_d}{35+t_d}\,\varepsilon_{sh}(u) \qquad \text{(A2.4)}$$

where

$\varepsilon_{sh}(t)$ = time related shrinkage strain
$\varepsilon_{sh}(u)$ = ultimate shrinkage (mircostrain)
t_d = age of concrete from onset of drying

and ultimate shrinkage is defined as

$$\varepsilon_{sh}(u) = \varepsilon_{nom}\, k_{v/s}\, k_{\lambda}\, k_{\xi}\, k_{\psi}\, k_{\alpha}\, k_{\tau}\, k_{\omega} \qquad \text{(A2.5)}$$

where

ε_{nom} nominal creep usually 780 mircostrain

$k_{v/s} = 1.2\, e^{-0.00472 v/s}$

$k_{\lambda} = 1.4 - 0.01\ \lambda$

$k_{\xi} = 0.89 + 0.00161\ \xi$

$k_{\psi} = 0.9 + 0.002\ \psi$

$k_{\alpha} = 0.95 + .008\ \alpha$

k_{τ} = nominal 1.0 for 7 days curing period (1.1 for 3 days)

$k_{\omega} = 0.75 + 0.00061\ \omega$

Basic creep coefficient is defined as

$$\phi_t = \frac{(t-s)^{0.6}}{10+(t-s)^{0.6}}\ \phi_u \qquad \text{(A2.6)}$$

where

ϕ_t = time related creep coefficient
ϕ_u = ultimate creep
t = total age of concrete
s = age at loading

and ultimate creep is defined as

$$\phi_u = \phi_{nom}\, k_s\, k_{v/s}\, k_{\lambda}\, k_{\xi}\, k_{\psi}\, k_{\alpha} \qquad \text{(A2.7)}$$

where

ϕ_{nom} nominal creep usually 2.35

$k_s = 1.25\, s^{-0.118}$ where s is the time of loading

$k_{v/s} = \frac{2}{3}\left(1 + 1.13\, e^{-0.0213 v/s}\right)$

$k_{\lambda} = 1.27 - 0.0067\ \lambda$

$k_{\xi} = 0.82 + 0.00264\ \xi$

$k_{\psi} = 0.88 + 0.0024\ \psi$

$k_{\alpha} = 0.46 + .09\ \alpha$

A 2.4 Age-Adjusted Effective Modulus

The constitutive equation for strain can be expressed as

$$\varepsilon(t) - \varepsilon_{sh}(t) = \alpha + \beta\, \phi(t,s) \quad \text{for } t > s > 0 \qquad \text{(Bazant 1972)}^6 \qquad \text{(A2.8)}$$

where

$\varepsilon(t)$ = total strain at (t) days from casting of concrete
$\varepsilon_{sh}(t)$ = total shrinkage strain at (t) days from casting of concrete
α , β = are constants
ϕ (t,s) = creep coefficient, where (s) is the time at load application and t > s

now for t = s (at the instant load is applied) (A2.8) becomes:

$$\varepsilon(s) - \varepsilon_{sh}(s) = \alpha \qquad \text{(A2.9)}$$

or

$$\varepsilon(s) = \alpha + \varepsilon_{sh}(s) \qquad \text{(A2.10)}$$

Therefore the total strain at the first application of load is the shrinkage strain plus α, therefore α is the instantaneous strain associated with the application of load at time (s), better know as the elastic strain.

The constitutive equations for stress (Bazant 1972)

$$\Delta\sigma(t) = \sigma_c(t) - \sigma_c(s) \qquad \text{(A2.11)}$$

where

$\Delta\sigma(t)$ = difference in stress over time after application of load
$\sigma_c(t)$ = total stress in concrete at time (t)
$\sigma_c(s)$ = initial stress at application of load

In effect $\Delta\sigma(t)$ is the difference in elastic stress from (s) to (t) which can be expressed in terms of the modulus of elasticity and strain. Bazant defines the age-adjusted modulus of elasticity to allow for experimental data showing the aging of concrete for creep.

Thus (A2.11) can be written as

$$\Delta\sigma(t) = E''(t,s)\,[\varepsilon(t) - \varepsilon(s) - \Delta\varepsilon''(t)] \text{ (Bazant 1972)} \qquad \text{(A2.12)}$$

where

$E''(t,s)$ = age-adjusted effective modulus and is given by (A2.13)
$\Delta\varepsilon''(t)$ = is the difference in creep and shrinkage strains from (s) to (t)

---

[6] Some symbols have been altered to reflect the nomenclature used in this report

The age-adjusted effective modulus as defined by Trost-Bazant (1967-1972) is

$$E''(t,s) = \frac{E_c(s)}{1 + \chi(t,s)\,\phi(t,s)} \qquad \text{(A2.13)}$$

where

$E_c(s)$ = modulus for the concrete section at the application of load
$\chi(t,s)$ = aging coefficient
all other symbols have the same meaning as defined previously

and the difference in concrete creep and shrinkage strain $\Delta\varepsilon''(t)$ is

$$\Delta\varepsilon''(t) = \frac{\sigma_c(s)}{E_c(s)}\phi(t,s) + \varepsilon_{sh}(t) - \varepsilon_{sh}(s) \quad \text{(Bazant 1972)} \qquad \text{(A2.14)}$$

where all symbols have the same meaning as defined previously.
Substituting into (A2.12)

$$\Delta\sigma(t) = \frac{E_c(s)}{1+\chi(t,s)\,\phi(t,s)}\left[\varepsilon(t) - \varepsilon(s) - \left[\frac{\sigma_c(s)}{E_c(s)}\phi(t,s) + \varepsilon_{sh}(t) - \varepsilon_{sh}(s)\right]\right] \qquad \text{(A2.15)}$$

rearranging

$$\Delta\sigma(t)\frac{1+\chi(t,s)\,\phi(t,s)}{E_c(s)} = \left[\varepsilon(t) - \frac{\sigma_c(s)}{E_c(s)}\phi(t,s) - \varepsilon_{sh}(t) - [\varepsilon(s) - \varepsilon_{sh}(s)]\right] \qquad \text{(A2.16)}$$

recalling (A2.9)

$$\varepsilon(s) - \varepsilon_{sh}(s) = \alpha \qquad \text{(A2.9)}$$

and α is the instantaneous axial strain at time (s) thus

$$\varepsilon(s) - \varepsilon_{sh}(s) = \alpha = \frac{\sigma_c(s)}{E_c(s)} \qquad \text{(A2.17)}$$

therefore (A2.16) becomes

$$\Delta\sigma(t)\frac{1+\chi(t,s)\,\phi(t,s)}{E_c(s)} = \left[\varepsilon(t) - \frac{\sigma_c(s)}{E_c(s)}(1+\phi(t,s)) - \varepsilon_{sh}(t)\right] \qquad \text{(A2.18)}$$

finally by rearranging (A2.18)

$$\varepsilon(t) = \frac{\sigma_c(s)}{E_c(s)}\left(1 + \phi(t,s)\right) + \varepsilon_{sh}(t) + \Delta\sigma(t)\,\frac{1 + \chi(t,s)\,\phi(t,s)}{E_c(s)} \qquad \text{(A2.19)}$$

Similar to Bazant's age adjusted effective modulus E' '(t,s), Faber (1927) defined a simple effective modulus as

$$E'(t,s) = \frac{E_c(s)}{1 + \phi(t,s)} \qquad \text{(A2.20)}$$

therefore by (A2.9), (A2.13), and (A2.20) becomes

$$\varepsilon(t) = \frac{\sigma_c(s)}{E'(t,s)} + \varepsilon_{sh}(t) + \frac{\Delta\sigma(t)}{E''(t,s)} \qquad \text{(A2.21)}$$

A2.5 Reinforced Concrete Model

To assist with the derivation consider the following (Beasley 1987)

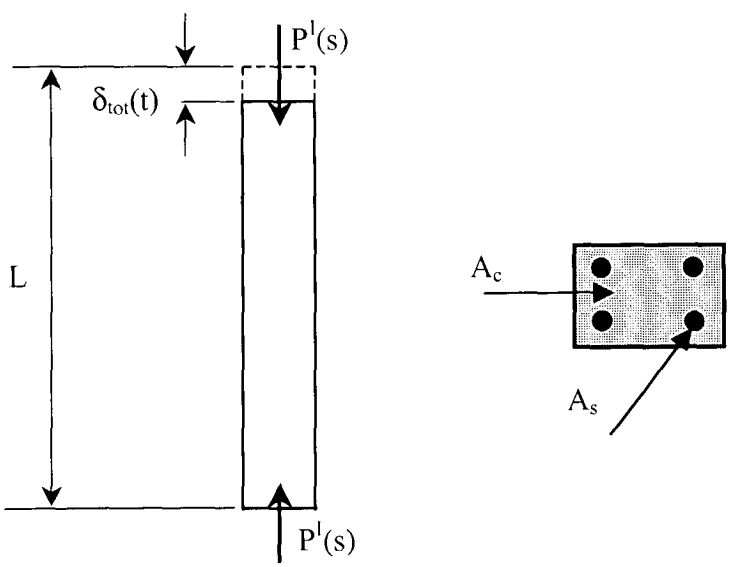


Figure A2.1 Axially loaded reinforced concrete column

where

$P'(s)$ = applied load at time (s), assumed uniform across entire section
A_c = area of concrete
A_s = area of steel
$\delta_{tot}(t)$ = total time dependant axial shortening of the column at time (t)
L = total effective length of the column with constant axial load P'

By assumption, concrete strain is equivalent to steel strain thus

$$\varepsilon(t) = \varepsilon_{st}(t) \qquad \text{(A2.22)}$$

where

$\varepsilon(t)$ = time dependent total strain in the member
$\varepsilon_{st}(t)$ = time dependent total steel strain

By equilibrium the axial load $P^l(s)$ must be resisted by internal stresses within the column thus

$$P^l(s) = \sigma_c(t)\, A_c + \sigma_s(t)\, A_s \qquad \text{(A2.23)}$$

where

$\sigma_c(t)$ = total stress in the concrete at time (t)
$\sigma_s(t)$ = total stress in the steel at time (t)

By rearranging (A2.10) the concrete stress in a reinforced concrete section can be expressed as the sum of the initial stress and the change in stress associated with concrete relaxation

$$\sigma_c(t) = \sigma_c(s) + \Delta\sigma(t) \qquad \text{(A2.24)}$$

where

$\sigma_c(s)$ = initial stress at time of loading (s)
$\Delta\sigma(t)$ = difference in stress over time after application of load

(A2.24) in (A2.23) and rearranging

$$\sigma_s(t) = \frac{1}{A_s}\left(P^l(s) - (\sigma_c(s) + \Delta\sigma(t))\, A_c\right) \qquad \text{(A2.25)}$$

For steel stress less than yield

$$\sigma_s(t) = \varepsilon_{st}(t)\, E_s \qquad \text{(A2.26)}$$

where

E_s = modulus of elasticity for steel

thus by (A2.22) , (A2.25) and (A2.26)

$$\varepsilon(t)\, E_s = \frac{1}{A_s}(P'(s) - (\sigma_c(s) + \Delta\sigma(t))\, A_c) \qquad \text{(A2.27)}$$

Recalling (A2.21)

$$\varepsilon(t) = \frac{\sigma_c(s)}{E'(t,s)} + \varepsilon_{sh}(t) + \frac{\Delta\sigma(t)}{E''(t,s)} \qquad \text{(A2.21)}$$

then by (A2.27) and (A2.21)

$$\left[\frac{\sigma_c(s)}{E'(t,s)} + \varepsilon_{sh}(t) + \frac{\Delta\sigma(t)}{E''(t,s)}\right] E_s A_s = (P'(s) - (\sigma_c(s) + \Delta\sigma(t))\, A_c) \qquad \text{(A2.28)}$$

If we define $n'(t,s)$ as the effective modular ratio and $n''(t,s)$as the age-adjusted effective modular ratio where

$$n'(t,s) = \frac{E_s}{E'(t,s)} \qquad \text{and} \qquad n''(t,s) = \frac{E_s}{E''(t,s)}$$

then rearranging (A2.28)

$$\Delta\sigma(t) = \frac{1}{1 + n''(t,s)\, p}\left[\frac{P'(s)}{A_c} - \sigma_c(s)(1 + n'(t,s)\, p) - \varepsilon_{sh}(t) E_s p\right] \qquad \text{(A2.29)}$$

where

$$p = \frac{A_s}{A_c}$$

By elastic analysis of the column at the time of load application the initial stress $\sigma_c(s)$ is

$$\sigma_c(s) = \frac{P'(s)}{A_c(1 + p\, n(s))} \qquad \text{(A2.30)}$$

where n(s) is the modular ration at time of load application and

$$n(s) = \frac{E_s}{E_c(s)} \qquad \text{(A2.31)}$$

By a final substitution of (A2.30)and (A2.29) into (A2.21)

$$\varepsilon(t) = \frac{P^{I}(s)}{A_c(1+n(s)p)\,E'(t,s)} + \varepsilon_{sh}(t) + \frac{1}{(1+n''(t,s)p)\,E''(t,s)}\left[\frac{P^{I}(s)}{A_c} - \frac{P^{I}(s)(1+n'(t,s)p)}{A_c(1+n(s)p)} - \varepsilon_{sh}(t)E_s p\right] \quad (A2.32)$$

By the definition of strain $\delta(t) = \varepsilon(t)\,L$ and rearranging (A2.32)

$$\delta(t) = \underbrace{\frac{P^{I}(s)L}{A_c(1+n(s)p)}\left[\frac{1}{E'(t,s)} + \frac{p\,(n(s)-n'(t,s))}{(1+n''(t,s)p)\,E''(t,s)}\right]}_{A(t,s)} + \underbrace{\varepsilon_{sh}(t)L}_{B(t)} - \underbrace{\frac{\varepsilon_{sh}(t)E_s pL}{(1+n''(t,s)p)\,E''(t,s)}}_{C(t)} \quad (A2.33)$$

where

A(t,s) represents the discrete elastic and creep component of axial shortening ($\delta_e(s) + \delta_c(t,s)$)

B(t) represents the discrete shrinkage component of axial shortening ($\delta_s(t)$)

C(t) represents the associated discrete reinforcing stiffening effect ($\delta_r(t)$)

Chapter 3

FRAMING

3.1 Introduction

Due to both structural and architectural requirements of TCBs, structural connections between horizontal and vertical elements are necessary. The design of connections will result in full moment transfer, partial moment transfer or no moment transfer[1] between the connected elements. Refer Figure 3.1. Providing a connection with either full moment transfer or partial moment transfer between connecting elements increases the complexity involved when determining internal force distribution and axial deformations in TCBs. In such cases, elastic, creep and shrinkage deformations associated with static loading are shared, thus a method for determining axial deformations for rigid[2] connections is required.

Although axial deformations make up only one part of the overall response to static load, when considering differential shortening it is only these actions that are of interest. Whilst techniques have been developed to allow isolated elements to be analysed for axial creep and shrinkage deformations, the introduction of full moment connections and framing between isolated elements further complicates the analysis of creep and shrinkage deformations.

The problem of axial shortening deformations with rigid connections is addressed in this chapter by firstly considering the nature of connections and design approaches of current typical construction practice and some of the approaches made by others in this area. A new procedure that allows complete solutions to be generated for entire structures is then formulated.

---

[1] Pin connections are typical connections without moment transfer
[2] Rigid connections are those with some form of moment transfer between elements

3.2 Review of Current Procedures

3.2.1 Connections

Currently under construction in Australia are a number of TCBs. Although some are of composite construction, the majority are reinforced concrete. In order to reduce cost and weight and to improve strength and durability, high strength concretes are often used in TCBs (Shilstone 1994). This allows the cross section of a column to be reduced. One of the unfortunate effects of reducing column area is that axial shortening increases (Beasley 1997). Controlling axial shortening can be achieved at the design stage by ensuring connections between columns and beams are rigid (Placzek 1997).

The World Tower building in Sydney will be the tallest residential concrete structure in Australia (Martin 2003). The tower is being constructed by a slip form system comprising of two storey lifts at a time. The structural frame is to be created by reinforced concrete columns and floors incorporating band beams of a similar design to that detailed below.

The majority of the information obtained on current connection details and building cycles is obtained from two major construction companies in Australia (Martin 2003). It is based on current projects either under construction or at the design stage from each company.

The information provided includes moment distributions of structures with basic gravity loads dominating and moment distributions for structures experiencing significant wind pressures and earthquakes. The later group of structures are designed for a moment envelope that includes load reversal. Differential axial shortening of framed structures experience load sharing and load reversal in a similar fashion as discussed previously in Section 2.4.

Design of connections for such cases is based on the moment envelopes with load reversal, refer Figure 3.1 below.

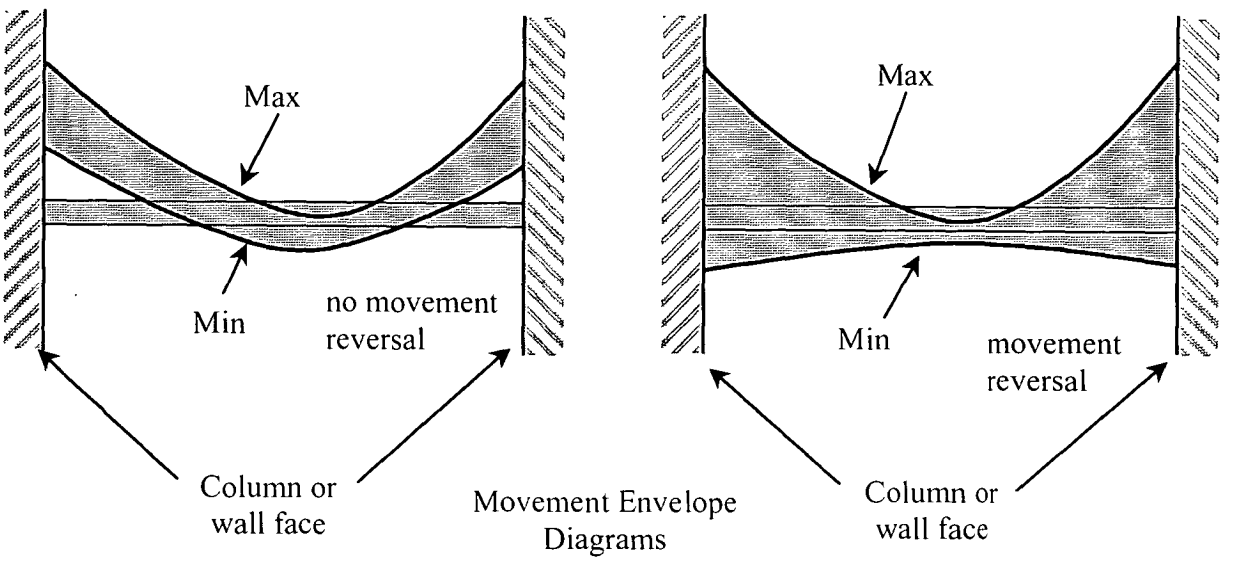


Figure 3.1 Sketches of moment envelopes for common connection design (Placzek 1997)

Figure 3.1 provides an insight into the current construction practices used in multi-storey construction. Structures without seismic or significant wind loads are generally designed for partial moment development of rigid connections without a capacity to withstand moment reversal. However, if an axial shortening analysis reveals a differential shortening problem, full development connection for moment reversal is usually adopted (Placzek 1997). This encourages load reversal and distributes axial forces.

In cases where differential axial deformations are identified, the column/core and beam connections are reinforced as continuous members, intersecting at each junction. In these cases the assumption of no joint rotation is important. In practice, minor rotation may occur, however, load redistribution is essentially due to shear transfer and is based on the stiffness relationship of the connecting member and will be equal and opposite at the two connected columns. Appendix A3.1 shows standard detailed joints of both types of joint design.

3.2.1.1 Composite construction

Increasingly, to conserve weight and the costs associated with the formwork required with insitu concrete, multi-storey structures are built using framed steel beams or precast elements (Martin 2003) that are assembled on site with insitu concrete limited to columns and floors. Such a method of construction may reduce the rigidity of framed connections. In these situations only, partially rigid or even pin connections need to be considered when determining framing action effects.

One of the precast concrete products that is gaining in popularity is Ultrafloor (Mendis et al 1999). This system consists of a reinforced precast beam and hardiform formboard to construct a platform. Refer Figure 3.2. Insitu concrete is then poured over the precast floor system locking the structure together. Eight systems are currently available for different load conditions and spans. In such systems all connections are pin based. The system is ideal where formwork is difficult or there are no special requirements for rigid connections.

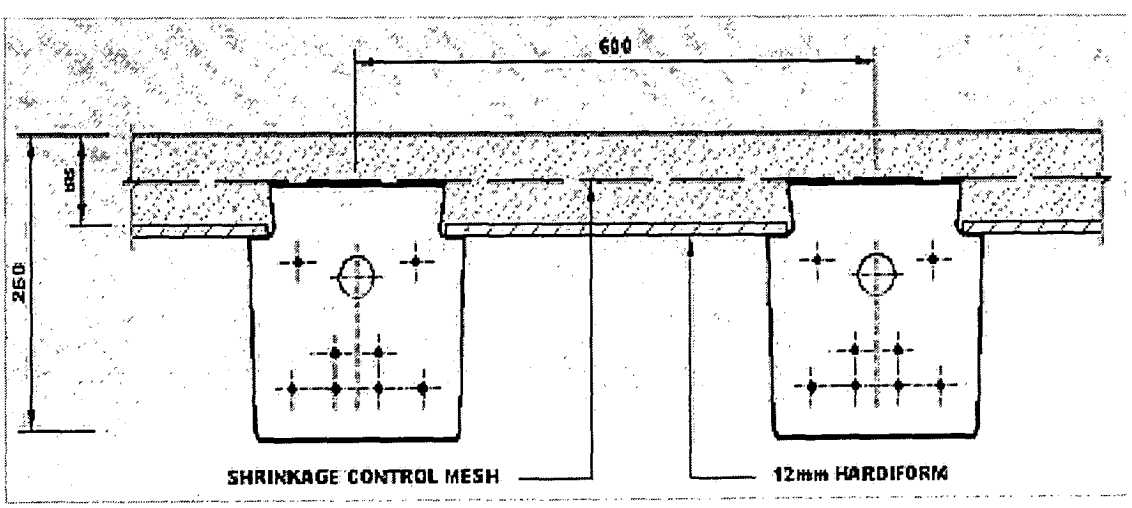


Figure 3.2 Section from Ultrafloor Website of 250C/400 UltraFloor System

The selection of a construction method is predominantly based on cost (Beasley 1997). In Australia a large proportion of TCBs are still fully constructed from reinforced concrete. Composite beam construction such as Ultrafloor are also becoming more widespread. Both methods of construction are difficult to analyse using conventional stiffness matrix methods as analysis of connections is limited to isotropic beams with either full or zero fixity. In order to increase the applicability of research into this area to account for plates and semi rigid connections, a general approach to handle stiffness at each connection is necessary. Instead of automatically applying the standard stiffness relationship, the application of the theory is extended to allow any stiffness value to be used and the program structured so the user could manipulate stiffness relationships at all connections.

This also allows an analysis of plates and band beams. Future expansion of the application to include cross bracing of multiple floors or plate analysis is possible. Although this program does not directly perform plate analysis, by force displacement analysis of flat plates with stiffened beams, a stiffness relationship between connections can be calculated and input by the user. Extending the program to incorporate existing plate analysis methods (Melerski 1991) would greatly enhance the useability of the program.

3.2.2 Research by Warner

In 1975 R.F. Warner (1975) published a report on the effect of beam interaction on concrete column shortening. At that time the report was one of the first on a definitive approach to beam/column interaction and is directly related to the subject at hand. The report outlines the mechanics of axial shortening and beam interaction for two framed columns.

Warner derives equations in a two-dimensional system for two columns and one beam. The main emphasis is on calculating the axial deformations of only two columns and

without the use of computers the method would have been cumbersome if applied to a large number of columns of many storeys. However the explanation of the structural system of a single storey system to multi-storey structures is still applicable and as a result the report provides the foundation for this research with respect to the mechanics of framing action in multi-storey buildings. Refer Figure 3.3

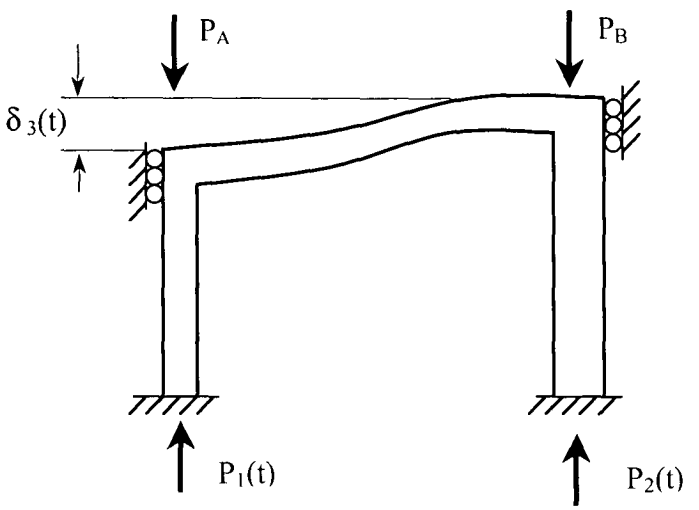


Figure 3.3 Simplified structural system for framed structures as proposed by Warner (1975)

where

$P_1(t)$ = internal axial force in Member 1
$P_2(t)$ = internal axial force in Member 2
P_A = constant applied load at Member 1
P_B = constant applied load at Member 2
$\delta_3(t)$ = differential shortening ($\delta_1(t) - \delta_2(t)$)

3.2.3 Research by Fintel and Khan

M.Fintel and F.R.Khan have published a number of research papers on the axial deformations of columns in tall structures (1971-1986). In the mid 1980s with the

development of more powerful computers, Fintel (1986) also showed the effects of axial deformations with connecting floor slabs. In the paper he describes in brief his methods for allowing for framing action effects. Essentially this involves calculating the unrestrained elastic and inelastic differential shortening, and by the use of a frame analysis program the framing effects can be simulated using an equivalent temperature drop for the unrestrained shortening values. While the method will allow for a modification of total axial deformations, the accuracy of the results depend on how well the framing program models a multi-storey building. Additionally the application of the simulated temperature drop is a source of possible error. The method also requires considerable computational effort with obvious consequences of numerical error.

Beasley (1997) and Placzek (1998) also adopt a similar method for predicting axial shortening of framed structures. This method is calibrated against actual projects[3] and has proved to be quite accurate, however, with approaches that attempt to simulate framing action there are more computations required and the designer needs to be meticulous in the application of the method. The potential for numerical error is therefore higher than adopting a method that already considers framing action when calculating internal loads.

3.3 Development of New Procedure

3.3.1 Introduction

Although a number of attempts have been made to quantify creep and shrinkage deformations in actual TCBs making allowances for rigid connections and frames between floors, there does not appear to be an adequate solution available that will produce a complete set of axial deformations for an entire structure. This chapter addresses all of the variables associated with developing such a procedure. The method is developed around three steps as follows;

[3] Emirates Tower, UAE

i) analysis of modern connection details, the building cycle and loading patterns

ii) consideration of the variables associated with creep and shrinkage models discussed in Chapter 2 and developing constitutive equations for elastic, creep and shrinkage displacements for vertically loaded elements in the structure

iii) generating a method of analysis that allows axial forces and displacements to be calculated for each element based on the structural system proposed by Warner (1975). See Figure 3.3

A number of assumptions and simplifications must be made in order to establish the appropriate models. The validity and associated errors are discussed in the following section.

3.3.2 Assumptions and methodology

The clarification and discussion on the assumptions made when formulating this new procedure only relate to the development of the model used to determine internal axial forces. The assumptions made relating to all other elements of the procedure are simply restated. Further clarification can be obtained in previous chapters.

The main assumptions and associated inaccuracies in summary are;

i) assumption of no joint rotation (refer Figure 3.4, 3.6, 3.7)
ii) stiffness relationship between columns
iii) superposition of load reversal (refer Figure 2.2)
iv) sum of large number of parts
v) iteration limitations
vi) variations in concrete properties
vii) creep and shrinkage models (refer Chapter 2.3)
viii) internal forces are assumed to be planar, axial and constant for the entire element

These assumptions are discussed in detail below;

The first and most important assumption is the absence of joint rotation (i). This greatly reduces the degrees of freedom and, whilst it limits the application of the theory to those structures that contain massive core elements, in general it is these structures that primarily exhibit large differential axial shortening problems. All tall reinforced concrete structures that experience the creep and shrinkage deformations predominately contain core elements that make this assumption valid. This structural system adopted by Warner (1975) best models TCBs.

The stiffness relation between columns (ii) is assumed to be for a full moment connection between the column and beam. Although this would suggest that the relationship is equal and opposite, in practice this would not be the case. The method is developed so that the stiffness relationship can be altered for each column. In application this would require that the engineer using the procedure be experienced enough to make sound decisions as to the actual rigidity of each connection.

The principle of superposition (iii) is well documented when applied to an monotonically increasing increment of axial creep. However, in the case of a framed structure there exists the likelihood that some elements will have reducing creep values. This in essence is the same as that for an element with load reversal. McHenry (1943) published research on such a case and concluded that

> *'with the removal of an axial load the concrete element experiences negative creep. However the rate of negative creep does not follow the same curve for that of an equivalent positive creep'.*

To simplify the development of the method it is assumed that both positive and negative creep both follow the same curves. The likely error associated with this assumption is small. The error when calculating the amount of creep is very small as the predominant creep values are positive, and as creep is only one of the four components of the total axial deformation, the total error is even less.

The process of calculating the total axial deformations involves the addition of a large number of small values (iv). This also includes the difference of two values that may be similar in size and thus a small error in either value could mean that the resulting difference is positive or negative. An error in magnitude could effect the distribution of forces between framed members. Such an error is very difficult to contain, so it is important to ensure that every small part is as accurate as possible.

The method for evaluating second order effects such as creep and shrinkage requires a process of iteration (v). With all iteration processes there must be a compromise made between accuracy and efficiency. The iteration method used in this research is basic yet effective. It is considered satisfactory as the emphasis is more on the actual results rather than the latest numerical methods.

As with all physical objects, the properties of the material needs to be evaluated (vi). One can only assume that the properties are constant through the entire cross-section. The actual values and the assumption of constant cross-section are both sources of error. In order to determine the extent of variation due to different concrete properties, we can make a statistical analysis of a structure within certain confidence intervals for each property, thus obtaining an expected range of axial deformations.

As with variations in concrete properties, the actual creep and shrinkage models (vii) are a source of error. Research has been done on a number of creep and shrinkage models (Koutsoukis and Beasley 1995) to determine which model best determines creep and shrinkage with the least amount of statistical variance. This research makes an attempt to determine a range of results but statistical variance is not of primary concern in this research.

Like most framing programs relying on stiffness matrix analysis, internal forces are assumed to be planar, axial and constant (viii) for the entire element (Melerski 1995). In the case of a large element such as a core element, the assumption that the axial force is constant over the entire cross section may not hold. The actual error caused by such an assumption is hard to quantify, but is expected to be small. In most cases the assumption is valid.

3.3.3 Stiffness matrix development

The general stiffness matrix developed below is obtained by reducing the complexity of force distribution by isolating the forces of concern. A precise mechanism that relates these internal forces with associated displacements is developed by the use of a stiffness matrix approach (Melerski 1995).

The two simplifications made in the generation of the stiffness matrix are

i) all framing elements considered are assumed to be horizontal, that is they constitute floors and beams only
ii) the stiffness multipliers associated with the framing elements are assumed to be equal at each end

Both of these simplifications are made to reduce the complexity of the derivation of the stiffness matrix. In the case of the second simplification the program written around the stiffness matrix allows for the stiffness relationship to differ at each end of the connecting member.

Beginning with one column on one arbitrary floor (j) we can develop the equation for the differential displacement between two columns over the entire height of the structure. By developing the force deflection relationship for the structure and applying the equation obtained for the differential displacement of the two columns, a simplified stiffness matrix is generated. The simplified stiffness matrix allows for the calculation of internal forces and differential displacements of framed column pairs for the entire structure at any point in time. The process relies heavily on the principle of superposition as discussed previously.

3.3.3.1 Constitutive equations for vertical elements

Figure 3.4 shows a pair of columns on the j^{th} storey between the j^{th} and j^{th}-1 floor. The idealised displacements of each column are defined and the desired nomenclature introduced.

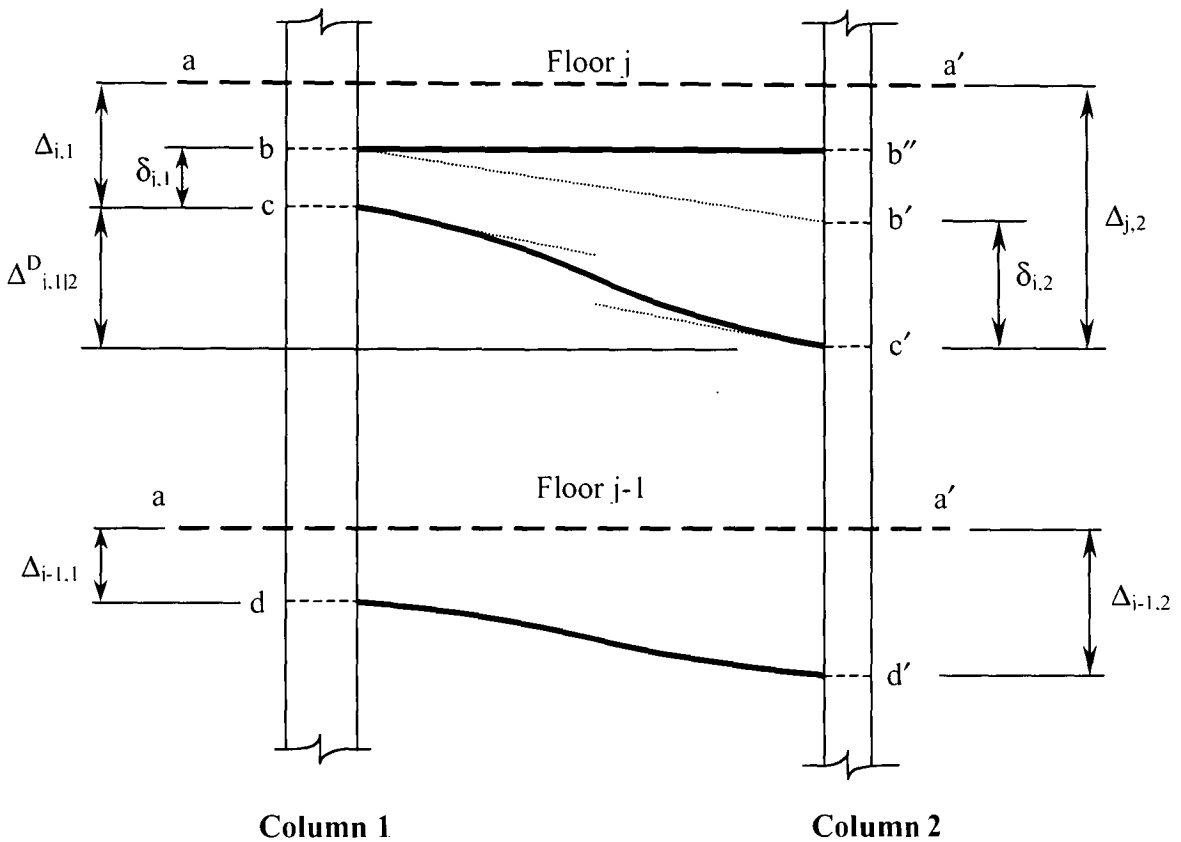


Figure 3.4 Typical column pair showing axial displacements

$\delta_{j,1}$, $\delta_{j,2}$ incremental time-varying axial shortening of columns 1 and 2 at the j^{th} floor due to internal axial loading within these columns at N days after completion

$\Delta_{j,1}$, $\Delta_{j,2}$ total time-varying axial shortening values of columns 1 and 2 at the j^{th} floor

$\Delta^{D}_{j,\,1|2}$ differential axial shortening value of the column pair 1 and 2 for the j^{th} storey

a - a' represents a zero deformation reference for columns built strictly to length

b - b' represents the column heights at the time of framing for columns built strictly to length

b - b'' represents the horizontal framing reference line at the time of framing

c - c' represents the column heights for columns 1 and 2 on the j^{th} storey at N days after completion of a building with columns built strictly to length

d - d' represents both the column heights for columns 1 and 2 on the j^{th}-1 storey at N days after completion of a building with columns built strictly to length and the location of the bottom of columns 1 and 2 on the j^{th} storey

From Figure 3.4 the differential displacement of column 1 with respect to column 2 ($\Delta^{D}_{j,1|2}$) can be clearly written as

$$\Delta^{D}_{j,1|2} = \Delta_{j,1} - \Delta_{j,2} \tag{3.1}$$

Similarly we can define the differential displacement for column 2 with respect to column 1 ($\Delta^{D}_{j,12|1}$) as

$$\Delta^{D}_{j,2|1} = \Delta_{j,2} - \Delta_{j,1} \tag{3.2}$$

At the time of framing the j^{th} floor, column 1 is at point b, whilst at the j^{th}-1 floor column 1 is at point d. Thus the axial shortening of column 1 for the j^{th} storey can be expressed as

$$\Delta_{j,1} = \Delta_{j-1,1} + \delta_{j,1} \tag{3.3}$$

similarly

$$\Delta_{j,2} = \Delta_{j-1,2} + \delta_{j,2} \tag{3.4}$$

Therefore from (3.3)and (3.4), (3.1) and (3.2) can be written as

$$\Delta^{D}_{j,1|2} = \Delta_{j-1,1} + \delta_{j,1} - (\Delta_{j-1,2} + \delta_{j,2}) \tag{3.5}$$

$$\Delta^{D}_{j,2|1} = \Delta_{j-1,2} + \delta_{j,2} - (\Delta_{j-1,1} + \delta_{j,1}) \tag{3.6}$$

alternatively

$$\Delta^{D}_{j,1|2} = \Delta_{j-1,1} - \Delta_{j-1,2} + \delta_{j,1} - \delta_{j,2} \tag{3.7}$$

$$\Delta^{D}_{j,2|1} = \Delta_{j-1,2} - \Delta_{j-1,1} + \delta_{j,2} - \delta_{j,1} \tag{3.8}$$

$\Delta^{D}_{j,1|2}$ and $\Delta^{D}_{j,2|1}$ should therefore be equal and opposite.

By a direct extension of (3.1)

$$\Delta^{D}_{j+1,1|2} = \Delta_{j+1,1} - \Delta_{j+1,2} \tag{3.9}$$

and from (3.3) (generally for any column i)

$$\Delta_{j+1,i} = \Delta_{j,i} + \delta_{j+1,i} \tag{3.10}$$

$$= \Delta_{j-1,i} + \delta_{j,i} + \delta_{j+1,i} \tag{3.11}$$

by (3.11) in (3.9,) (3.7) and (3.8) can be written as

$$\Delta^{D}_{j+1,1|2} = \Delta_{j-1,1} - \Delta_{j-1,2} + \delta_{j,1} - \delta_{j,2} + \delta_{j+1,1} - \delta_{j+1,2} \tag{3.12}$$

$$\Delta^{D}_{j+1,2|1} = \Delta_{j-1,2} - \Delta_{j-1,1} + \delta_{j,1} - \delta_{n,2} + \delta_{j+1,1} - \delta_{j+1,2} \tag{3.13}$$

By extending (3.12) and (3.13) a series can be generated to express differential axial shortening for a column pair in terms of incremental time-varying axial shortening for each column

$$\Delta^{D}_{n,1|2} = \Delta_{0,1} - \Delta_{0,2} + \delta_{1,1} - \delta_{1,2} + \ldots + \delta_{j,1} - \delta_{j,2} + \ldots + \delta_{n,1} - \delta_{n,2} \tag{3.14}$$

$$\Delta^{D}_{n,2|1} = \Delta_{0,2} - \Delta_{0,1} + \delta_{1,2} - \delta_{1,1} + \ldots + \delta_{j,2} - \delta_{j,1} + \ldots + \delta_{n,2} - \delta_{n,1} \tag{3.15}$$

3.3.4 Determination of F_n - Δ_n relationship

Load deflection relationships are well known for a number of beam column relationships. In a simplified two-dimensional section of a TCB, assuming no rotation of the column, the basic stiffness relation enables us to calculate load/deflection relationships. This relationship is usually linear up until cracking in the case of concrete members, or yield in the case of steel members. The stiffness matrix developed below assumes a linear relationship between load and deflection, however it can easily be extended further to account for cracked concrete sections. Such an extension has been made by Warner (1975).

In order to develop the constitutive equations for the force-deflection relationship associated with differential axial shortening of column pairs, consider the node associated with a particular column at the j^{th} storey. We can evaluate the equilibrium of forces for all connecting members associated with this particular node. Refer Figure 3.5.

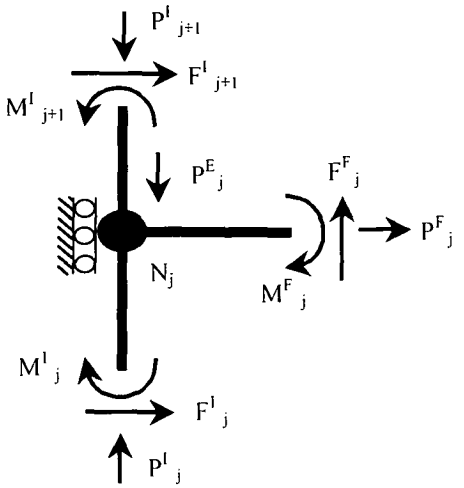


Figure 3.5 Isolated node on the j^{th} storey

where

The super-script I refers to internal, E refers to external and F refers to frame

N_j is the node at the jth floor. The node is restrained from rotation as per the assumptions discussed previously. Connecting to this node are any of the following:

(a) Vertical supports from the storey above (j+1)
(b) Vertical supports to the storey below
(c) Framing elements. The framing elements may be from any direction and may connect any other node to N_j. The basic stiffness matrix is derived for floor framing elements only and therefore all elements can be assumed to be at 90 degrees to vertical supports. Shown in Figure 3.5 is one element only

P^I_{j+1} is the internal axial force in the vertical support from the $j+1^{th}$ floor. This force is assumed to be planar, axial and constant for the entire element from the $j+1^{th}$ storey to the node at the j^{th} storey. Axial loads on the node are compressive

F^I_{j+1} is the internal shear force in the vertical support from the $j+1^{th}$ storey acting on the node at the j^{th} floor

M^I_{j+1} is the internal moment in the vertical support from the $j+1^{th}$ storey acting on the node at the j^{th} floor

P^I_j is the internal axial force in the vertical support of the j^{th} floor. This force is assumed to be planar, axial and constant for the entire element at the j^{th} storey to the node below. Axial loads on the node are compressive

F^I_j is the internal shear force in the vertical support of the j^{th} floor

M^I_{j+1} is the internal moment in the vertical support of the j^{th} floor

P^E_j is the external additional load applied to the node N_j. The force is the sum of all additional external forces at the j^{th} floor node. Such forces include all dead and live floor loads and any construction loading carried by vertical supports

P^F_j is the internal axial force in the framing element connected to node at the j^{th} floor. Axial loads on the node are compressive

F^F_j is the internal axial force resulting from shear produced by the framing element connected to node at the j^{th} floor

M^F_j is the internal moment in the framing element connected to node at the j^{th} floor

The node (N_j) is assumed to be fixed for rotation so all of the moments within the connecting members (M^F_j, M^I_j , M^I_{j+1}) cannot transfer past the node and into adjacent members. As we are not interested in the flexure of any individual element, Figure 3.5 can be simplified as below. Here all symbols have the same meaning as defined previously.

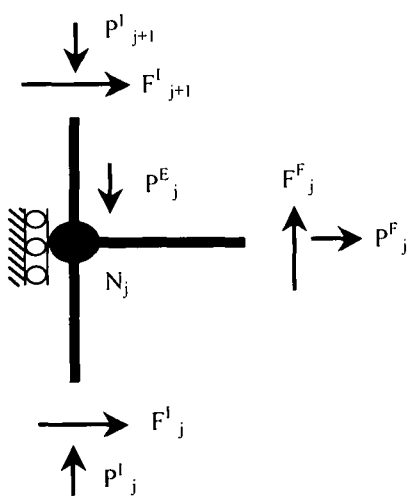


Figure 3.6 Modified isolated node on the j^{th} storey.
(Constituative equations for nodal forces are produced from this diagram)

By the sum of horizontal forces in Figure 3.6

$$F^I_j + F^I_{j+1} + P^F_j = 0 \tag{3.16}$$

and by the sum of vertical forces

$$P^I_j + F^F_j - P^I_{j+1} = P^E_j \tag{3.17}$$

rearranging (3.17)

$$P^I_j = P^I_{j+1} + P^E_j - F^F_j \tag{3.18}$$

(3.18) is the constitutive equation for nodal forces for the simplified model with only horizontal and vertical forces acting at nodes.

By considering a column pair and Figure 3.6 the axial forces associated with the axial displacements previously illustrated in Figure 3.4 can be defined. The line b - b' is removed to simplify the diagram.

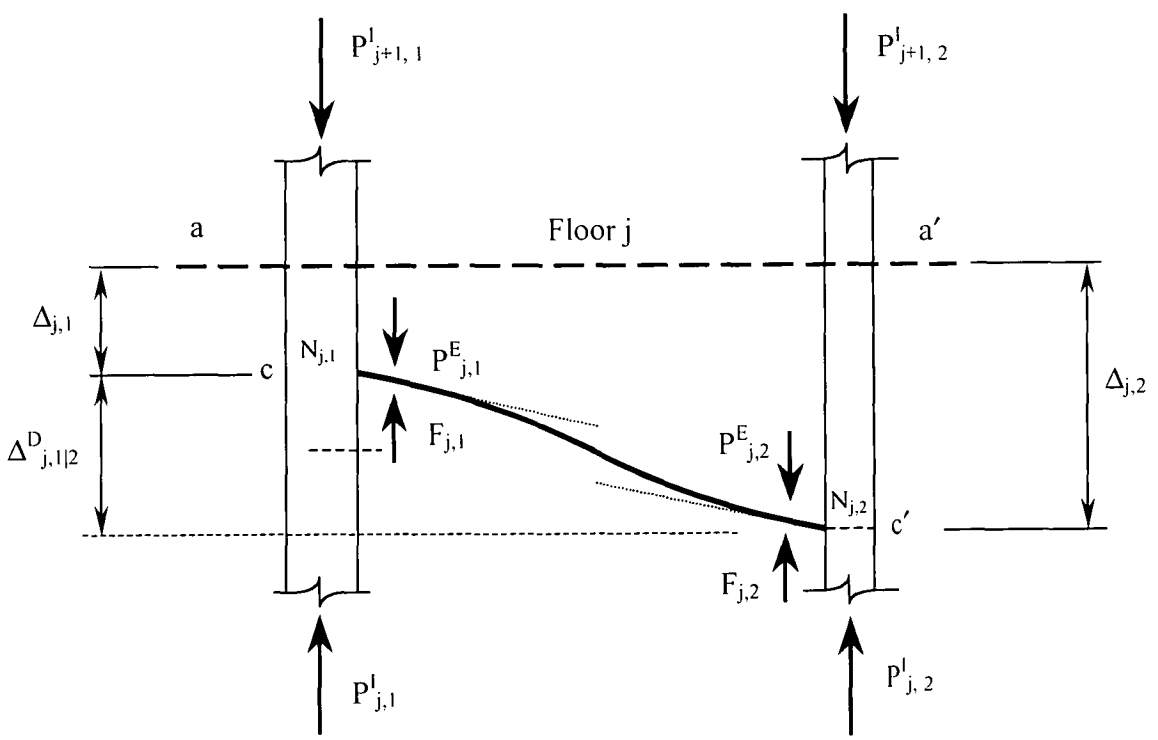


Figure 3.7 Typical column pair showing axial forces on nodes

$P^{I}_{j,1}$, $P^{I}_{j,2}$ these forces will be referred to as *resultant internal compression forces* and are the resultant internal forces in columns 1 and 2 supporting the j^{th} floor, from the equilibrium of nodal forces at the j^{th} floor. Internal forces are assumed to be planar, axial and constant for the entire element of the j^{th} storey below. Forces are positive if in compression

$P^{I}_{j+1,1}$, $P^{I}_{j+1,2}$ these forces will be referred to as *resultant internal compression forces* and are the internal forces in columns 1 and 2 supporting the $j+1^{th}$ floor from the equilibrium of forces at the $j+1^{th}$ floor. Internal forces are assumed to be planar, axial and constant for the entire element of the $j+1^{th}$ storey. Forces are positive if in compression

$P^{E}_{j,1}$, $P^{E}_{j,2}$ these forces will be referred to as *external forces* and are the sum of all additional external forces at the j^{th} floor nodes for columns 1 and 2. Such forces include all dead and live floor loads and any construction loading carried by column 1 or 2

$F_{j,1}$, $F_{j,2}$ these forces will be referred to as *framing forces* and are the internal forces in columns 1 and 2 at the j^{th} floor resulting from the transfer of shear forces from the framing member between columns 1 and 2.[4] Framing forces are the result of the differential vertical displacement ($\Delta^{D}_{j,1|2}$) and the continuity of the rigid connection between columns 1 and 2 at the j^{th} floor

All other symbols have the same meaning as defined previously.

For each column (3.18) can be written as

$$P^{I}_{j,1} = P^{I}_{j+1,1} + P^{E}_{j,1} - F_{j,1} \tag{3.19}$$

$$P^{I}_{j,2} = P^{I}_{j+1,2} + P^{E}_{j,2} - F_{j,2} \tag{3.20}$$

In order to determine the framing force associated with the rigid beam connection we define a stiffness multiplier such that

$$F_j = k_j \Delta^{D}_j \tag{3.21}$$

Where k_j is the known stiffness relationship between F_j and Δ^{D}_j and is assumed linear for the reasons discussed above.

[4] Note the superscript F is neglected as this is the only action from the framing element considered for the simplified case

By definition Δ^D_j represents the differential displacement of two conjoined elements, thus for any given element there may be more than one joining element. Reintroducing the nomenclature for two adjacent columns then

$$F_{j,1|2} = k_{j,1|2}\,\Delta^D_{j,1|2} \tag{3.22}$$

Here $k_{j,1|2}$ is the stiffness multiplier between columns 1 and 2. In the simplified matrix $k_{j,1|2}$ and $k_{j,2|1}$ are identical. However with different end connections it is possible for $k_{j,1|2}$ and $k_{j,2|1}$ to differ due to different degrees of rigidity.

For the two columns 1 and 2 at a general floor j, (3.19) and (3.20) can be written as

$$P^I_{j,1} = P^I_{j+1,1} + P^E_{j,1} - F_{j,1|2} \tag{3.23}$$ (column 1)

and

$$P^I_{j,2} = P^I_{j+1,2} + P^E_{j,2} - F_{j,2|1} \tag{3.24}$$ (column 2)

substituting (3.22)

$$P^I_{j,1} = P^I_{j+1,1} + P^E_{j,1} - k_{j,1|2}\,\Delta^D_{j,1|2} \tag{3.25}$$

and

$$P^I_{j,2} = P^I_{j+1,2} + P^E_{j,2} - k_{j,2|1}\,\Delta^D_{j,2|1} \tag{3.26}$$

Recall from (3.7) and (3.8) that

$$\Delta^D_{j,1|2} = \Delta_{j-1,1} - \Delta_{j-1,2} + \delta_{j,1} - \delta_{j,2} \tag{3.7}$$

$$\Delta^D_{j,2|1} = \Delta_{j-1,2} - \Delta_{j-1,1} + \delta_{j,2} - \delta_{j,1} \tag{3.8}$$

thus

$$P^I_{j,1} = P^I_{j+1,1} + P^E_{j,1} - k_{j,1|2}\,(\Delta_{j-1,1} - \Delta_{j-1,2} + \delta_{j,1} - \delta_{j,2}) \tag{3.27}$$

and

$$P^I_{j,2} = P^I_{j+1,2} + P^E_{j,2} - k_{j,2|1}\,(\Delta_{j-1,2} - \Delta_{j-1,1} + \delta_{j,2} - \delta_{j,1}) \tag{3.28}$$

In order to fully develop the stiffness matrix for a complete system we need to re-arrange (3.27) and (3.28) and then introduce the boundary conditions. By considering two columns and (n) storeys, then from Figure 3.8 we define the boundary conditions for a structure constructed on a rigid base.

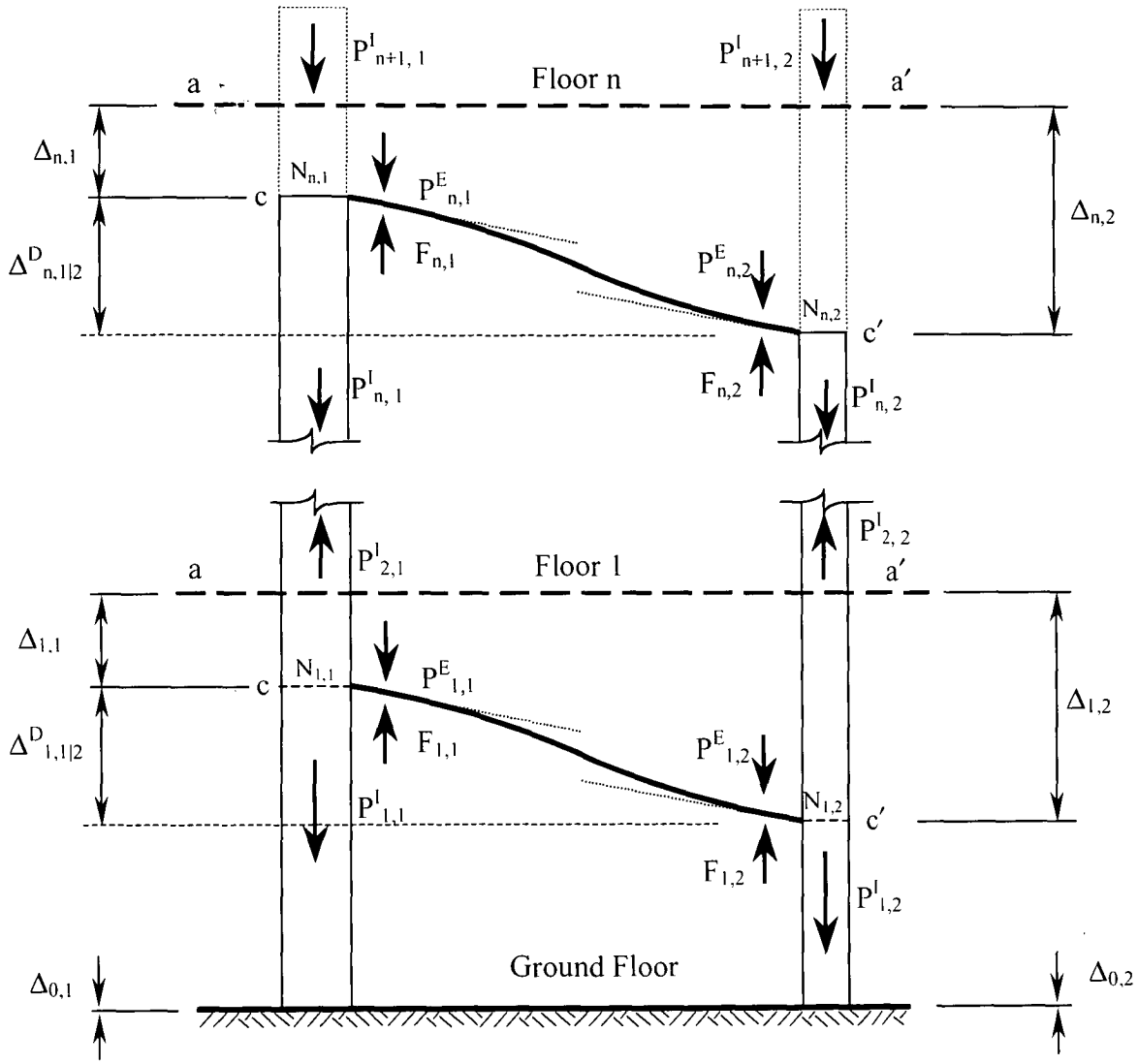


Figure 3.8 Typical column pairs, ground floor and top (nth) floor

From Figure 3.8

a) Lower boundary condition

$$\Delta_0 = 0, \qquad \text{(for all columns)}$$

where Δ_0 represents the deflection of the lowest floor (i.e. ground floor)

From Figure 3.8 and Figure 3.4

b) Upper boundary condition

$$P^{I}_{n+1} = 0 \quad \text{(for all columns)}$$

noting that (n) is the highest floor level and thus there are no columns above this storey.

Applying the lower boundary condition in (3.27) and (3.28) for j = 1 (first floor)

$$P^{I}_{1,1} - P^{I}_{2,1} = P^{E}_{1,1} - k_{1,1|2}(\delta_{1,1} - \delta_{1,2}) \tag{3.29}$$

and

$$P^{I}_{1,2} - P^{I}_{2,2} = P^{E}_{1,2} - k_{1,2|1}(\delta_{1,2} - \delta_{1,1}) \tag{3.30}$$

by (3.14) and (3.14), in (3.25) and (3.26) with the lower bound condition for the j^{th} floor

$$P^{I}_{j,1} - P^{I}_{j+1,1} = P^{E}_{j,1} - k_{j,1|2}(\delta_{1,1} - \delta_{1,2} + \ldots + \delta_{j,1} - \delta_{j,2}) \tag{3.31}$$

and

$$P^{I}_{j,2} - P^{I}_{j+1,2} = P^{E}_{j,2} - k_{j,2|1}(\delta_{1,2} - \delta_{1,1} + \ldots + \delta_{j,2} - \delta_{j,1}) \tag{3.32}$$

similarly with the upper boundary condition then for the n^{th} floor

$$P^{I}_{n,1} = P^{E}_{n,1} - k_{n,1|2}(\delta_{1,1} - \delta_{1,2} + \ldots + \delta_{j,1} - \delta_{j,2} + \ldots + \delta_{n,1} - \delta_{n,2}) \tag{3.33}$$

and

$$P^{I}_{n,2} = P^{E}_{n,2} - k_{n,2|1}(\delta_{1,2} - \delta_{1,1} + \ldots + \delta_{j,2} - \delta_{j,1} + \ldots + \delta_{n,2} - \delta_{n,1}) \tag{3.34}$$

Beginning with (3.26) and (3.27) in the generation of the structural stiffness matrix, then by expanding for the first storey we can write

$$\begin{array}{c} \\ 1^{st}\ \text{Storey} \\ 2^{nd} \\ \text{Storey} \end{array}\!\!\begin{array}{c} \begin{array}{cc} 1^{st}\ \text{Storey} & 2^{nd}\ \text{Storey} \end{array} \\ \left[\begin{array}{cc:cc} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ \hdashline & & & \end{array}\right] \end{array} \begin{bmatrix} P^{I}_{1,1} \\ P^{I}_{1,2} \\ \hdashline P^{I}_{2,1} \\ P^{I}_{2,2} \end{bmatrix} = \begin{bmatrix} P^{E}_{1,1} \\ P^{E}_{1,2} \\ \hdashline P^{E}_{2,1} \\ P^{E}_{2,2} \end{bmatrix} + \begin{array}{c} \begin{array}{cc} 1^{st}\ \text{Storey} & 2^{nd}\ \text{Storey} \end{array} \\ \left[\begin{array}{cc:cc} -k_{1,1/2} & k_{1,1/2} & 0 & 0 \\ k_{1,2/1} & -k_{1,2/1} & 0 & 0 \\ \hdashline & & & \end{array}\right] \end{array} \begin{bmatrix} \delta_{1,1} \\ \delta_{1,2} \\ \hdashline \delta_{2,1} \\ \delta_{2,2} \end{bmatrix}$$

Matrix equation 3.1

From Matrix equation 3.1 above $\begin{bmatrix} -k_{1,1/2} & k_{1,1/2} \\ k_{1,2/1} & -k_{1,2/1} \end{bmatrix}$ is defined as the stiffness matrix for two columns on the ground storey connected by framing elements[5]. For simplicity this will be referred to as $[k_1]_2$ where the subscript 2 refers to a 2 x 2 matrix and the subscript 1 refers to the first storey.

With further simplifications then Matrix equation 3.1 can be written as

$$1^{st}\text{Storey}\left[\begin{array}{c|c} [A]_2 & [-A]_2 \\ \hline [...] & [...] \end{array}\right]\left[\begin{array}{c} \{P^I{}_1\}_2 \\ \hline \{P^I{}_2\}_2 \end{array}\right] = \left[\begin{array}{c} \{P^E{}_1\}_2 \\ \hline \{P^E{}_2\}_2 \end{array}\right] + \left[\begin{array}{c|c} [k_1]_2 & [0] \\ \hline [...] & [...] \end{array}\right]\left[\begin{array}{c} \{\delta_1\}_2 \\ \hline \{\delta_2\}_2 \end{array}\right]$$

Matrix equation 3.2

where $[A]_2$ is the identity matrix for a 2 x 2 matrix

$\{P^I{}_n\}_2$ is the 2 x 1 column of internal forces at the nth storey

$\{P^E{}_n\}_2$ is the 2 x 1 column of external forces at the nth storey

$\{\delta_n\}_2$ is the 2 x 1 column of associated axial shortening values at the nth storey

Further rearranging of (3.31) to (3.34)

$$P^I{}_{j,1} - P^I{}_{j+1,1} = P^E{}_{j,1} - k_{j,1|2}(\delta_{1,1} + + \delta_{j,1} - (\delta_{1,2} + ... + \delta_{j,2})) \quad (3.35)$$

$$P^I{}_{j,2} - P^I{}_{j+1,2} = P^E{}_{j,2} - k_{j,2|1}(\delta_{1,2} + ... + \delta_{j,2} - (\delta_{1,1} + ... + \delta_{j,1})) \quad (3.36)$$

$$P^I{}_{n,1} = P^E{}_{n,1} - k_{n,1|2}(\delta_{1,1} + ... + \delta_{j,1} + ... + \delta_{n,1} - (\delta_{1,2} + ... + \delta_{j,2} + ... + \delta_{n,2})) \quad (3.37)$$

$$P^I{}_{n,2} = P^E{}_{n,2} - k_{n,2|1}(\delta_{1,2} + ... + \delta_{j,2} + ... + \delta_{n,2} - (\delta_{1,1} + ... + \delta_{j,1} + ... + \delta_{n,1})) \quad (3.38)$$

If we define $\delta\delta_{j,1} = \sum_{i=1}^{i=j} \delta_{i,1}$ then

$$P^I{}_{j,1} - P^I{}_{j+1,1} = P^E{}_{j,1} - k_{j,1|2}(\delta\delta_{j,1} - \delta\delta_{j,2}) \quad (3.39)$$

$$P^I{}_{j,2} - P^I{}_{j+1,2} = P^E{}_{j,2} - k_{j,2|1}(\delta\delta_{j,2} - \delta\delta_{j,1}) \quad (3.40)$$

$$P^I{}_{n,1} = P^E{}_{n,1} - k_{n,1|2}(\delta\delta_{n,1} - \delta\delta_{n,2}) \quad (3.41)$$

$$P^I{}_{n,2} = P^E{}_{n,2} - k_{n,2|1}(\delta\delta_{n,2} - \delta\delta_{n,1}) \quad (3.42)$$

---

[5] For a conventionally framed member $k_{1,1|2}$ is simply $\frac{12 E_{1|2} I_{1|2}}{\ell_{1|2}^{3}}$

and $\delta\delta_{1,1} = \sum_{i=1}^{i=1}\delta_{1,1} = \delta_{1,1}$

By (3.39) and (3.40) and with similar simplifications above, Matrix equation 3.1 at storey j becomes

$$j^{th}\ \text{Storey}\left[\begin{array}{c|c|c|c} [A]_2 & [-A]_2 & [...] & [...] \\ \hline [...] & [A]_2 & [-A]_2 & [...] \\ \hline [...] & [...] & [A]_2 & [-A]_2 \end{array}\right]\left[\begin{array}{c} \{P^I_{j-1}\}_2 \\ \hline \{P^I_j\}_2 \\ \hline \{P^I_{j+1}\}_2 \end{array}\right] = \left[\begin{array}{c} \{P^E_{j-1}\}_2 \\ \hline \{P^E_j\}_2 \\ \hline \{P^E_{j+1}\}_2 \end{array}\right] + \left[\begin{array}{c|c|c} [k_{j-1}]_2 & [0] & [0] \\ \hline [0] & [k_j]_2 & [0] \\ \hline [0] & [0] & [k_{j+1}]_2 \end{array}\right]\left[\begin{array}{c} \{\delta\delta_{j-1}\}_2 \\ \hline \{\delta\delta_j\}_2 \\ \hline \{\delta\delta_{j+1}\}_2 \end{array}\right]$$

Matrix equation 3.3

Likewise by (3.41) and (3.42) and with similar simplifications above, Matrix equation 3.1 at storey j becomes

$$n^{th}\ \text{Storey}\left[\begin{array}{c|c} [A]_2 & [-A]_2 \\ \hline [...] & [A]_2 \end{array}\right]\left[\begin{array}{c} \{P^I_{n-1}\}_2 \\ \hline \{P^I_n\}_2 \end{array}\right] = \left[\begin{array}{c} \{P^E_{n-1}\}_2 \\ \hline \{P^E_n\}_2 \end{array}\right] + \left[\begin{array}{c|c} [k_{n-1}]_2 & [0] \\ \hline [...] & [k_n]_2 \end{array}\right]\left[\begin{array}{c} \{\delta\delta_{n-1}\}_2 \\ \hline \{\delta\delta_n\}_2 \end{array}\right]$$

Matrix equation 3.4

Incorporating Matrix equations 3.2, 3.3, 3.4 and recalling that $\delta_{1,1} = \delta\delta_{1,1}$ then the general matrix equation for a two-column system can be written as

$$\left[\begin{array}{ccccccccc} [A]_2 & [-A]_2 & & & & & & & \\ & [A]_2 & [..]_2 & & & & & & \\ & & [..]_2 & [..]_2 & & & & & \\ & & & [A]_2 & [-A]_2 & & & & \\ & & & & [A]_2 & [..]_2 & & & \\ & & & & & [..]_2 & [..]_2 & & \\ & & & & & & [A]_2 & & \end{array}\right]\left[\begin{array}{c} \{P^I_1\}_2 \\ \{P^I_2\}_2 \\ \cdots \\ \{P^I_j\}_2 \\ \{P^I_{j+1}\}_2 \\ \cdots \\ \{P^I_n\}_2 \end{array}\right] = \left[\begin{array}{c} \{P^E_1\}_2 \\ \{P^E_2\}_2 \\ \cdots \\ \{P^E_j\}_2 \\ \{P^E_{j+1}\}_2 \\ \cdots \\ \{P^E_n\}_2 \end{array}\right] + \left[\begin{array}{ccccccc} [k_1]_2 & & & & & & \\ & [k_2]_2 & & & & & \\ & & [...]_2 & & & & \\ & & & [k_j]_2 & & & \\ & & & & [k_{j+1}]_2 & & \\ & & & & & [...]_2 & \\ & & & & & & [k_n]_2 \end{array}\right]\left[\begin{array}{c} \{\delta\delta_1\}_2 \\ \{\delta\delta_2\}_2 \\ \cdots \\ \{\delta\delta_j\}_2 \\ \{\delta\delta_{j+1}\}_2 \\ \cdots \\ \{\delta\delta_n\}_2 \end{array}\right]$$

Matrix equation 3.5

The complete matrix for a multi-storey structure of *n* storeys and *m* columns can then simply be written as

$$\begin{bmatrix} [A]_m & [-A]_m & & & & & & \\ & [A]_m & [..]_m & & & & & \\ & & [..]_m & [..]_m & & & & \\ & & & [A]_m & [-A]_m & & & \\ & & & & [A]_m & [..]_m & & \\ & & & & & [..]_m & [..]_m & \\ & & & & & & & [A]_m \end{bmatrix} \begin{Bmatrix} \{P^I_1\}_m \\ \{P^I_2\}_m \\ \dots \\ \{P^I_j\}_m \\ \{P^I_{j+1}\}_m \\ \dots \\ \{P^I_n\}_m \end{Bmatrix} = \begin{Bmatrix} \{P^E_1\}_m \\ \{P^E_2\}_m \\ \dots \\ \{P^E_j\}_m \\ \{P^E_{j+1}\}_m \\ \dots \\ \{P^E_n\}_m \end{Bmatrix} + \begin{bmatrix} [k_1]_m & & & & & & \\ & [k_2]_m & & & & & \\ & & [...]_m & & & & \\ & & & [k_j]_m & & & \\ & & & & [k_{j+1}]_m & & \\ & & & & & [...]_m & \\ & & & & & & [k_n]_m \end{bmatrix} \begin{Bmatrix} \{\delta\delta_1\}_m \\ \{\delta\delta_2\}_m \\ \dots \\ \{\delta\delta_j\}_m \\ \{\delta\delta_{j+1}\}_m \\ \dots \\ \{\delta\delta_n\}_m \end{Bmatrix}$$

General matrix equation 3.6

3.3.4.1 Stiffness matrix for multiple pairs of columns

The stiffness matrices at each storey need to be defined for the general matrix equation. Recalling that the 2 x 2 column arrangement $[k_1]_2 = \begin{bmatrix} -k_{1,1-2} & k_{1,1-2} \\ k_{1,1-2} & -k_{1,1-2} \end{bmatrix}$, which was obtained from (26) and (27). In general there is potential for framing between any two columns on one storey, therefore for each and every column pair on a particular storey we must define a corresponding framing stiffness relation, $k_{n,i|j}$. For each and every pair (i,j), $k_{n,i|j}$ is defined as the framing stiffness at column i due to the framing between i and j, and $k_{n,j|i}$ is defined as the framing stiffness at column j due to the framing between i and j.

When generating the general stiffness matrix for each storey, consider a particular system with four columns on each storey.

The four equations of equilibrium on the first storey are as follows;

$$P^{I}_{1,1} - P^{I}_{2,1} = P^{E}_{1,1} - (k_{1,1|2}\Delta_{1,1|2} + k_{1,1|3}\Delta_{1,1|3} + k_{1,1|4}\Delta_{1,1|4}) \quad (3.43)$$

$$P^{I}_{1,2} - P^{I}_{2,2} = P^{E}_{1,2} - (k_{1,2|1}\Delta_{1,2|1} + k_{1,2|3}\Delta_{1,2|3} + k_{1,2|4}\Delta_{1,2|4}) \quad (3.44)$$

$$P^{I}_{1,3} - P^{I}_{2,3} = P^{E}_{1,3} - (k_{1,3|1}\Delta_{1,3|1} + k_{1,3|2}\Delta_{1,3|2} + k_{1,3|4}\Delta_{1,3|4}) \quad (3.45)$$

$$P^{I}_{1,4} - P^{I}_{2,4} = P^{E}_{1,4} - (k_{1,4|1}\Delta_{1,4|1} + k_{1,4|2}\Delta_{1,4|2} + k_{1,4|3}\Delta_{1,4|3}) \quad (3.46)$$

Introducing the definitions of $\Delta^{D}_{n,i|j}$ and applying the lower boundary condition then

$$P^{I}_{1,1} - P^{I}_{2,1} = P^{E}_{1,1} - (k_{1,1|2}[\delta_{1,1} - \delta_{1,2}] + k_{1,1|3}[\delta_{1,1} - \delta_{1,3}] + k_{1,1|4}[\delta_{1,1} - \delta_{1,4}]) \quad (3.47)$$

$$P^{I}_{1,2} - P^{I}_{2,2} = P^{E}_{1,2} - (k_{1,2|1}[\delta_{1,2} - \delta_{1,1}] + k_{1,2|3}[\delta_{1,2} - \delta_{1,3}] + k_{1,2|4}[\delta_{1,2} - \delta_{1,4}]) \quad (3.48)$$

$$P^{I}_{1,3} - P^{I}_{2,3} = P^{E}_{1,3} - (k_{1,3|1}[\delta_{1,3} - \delta_{1,1}] + k_{1,3|2}[\delta_{1,3} - \delta_{1,2}] + k_{1,3|4}[\delta_{1,3} - \delta_{1,4}]) \quad (3.49)$$

$$P^{I}_{1,4} - P^{I}_{2,4} = P^{E}_{1,4} - (k_{1,4|1}[\delta_{1,4} - \delta_{1,1}] + k_{1,4|2}[\delta_{1,4} - \delta_{1,2}] + k_{1,4|3}[\delta_{1,4} - \delta_{1,3}]) \quad (3.50)$$

rearranging

$$P^{I}_{1,1} - P^{I}_{2,1} = P^{E}_{1,1} + (-\delta_{1,1}[k_{1,1|2} + k_{1,1|3} + k_{1,1|4}] + \delta_{1,2}k_{1,1|2} + \delta_{1,3}k_{1,1|3} + \delta_{1,4}k_{1,1|4}) \quad (3.51)$$

$$P^{I}_{1,2} - P^{I}_{2,2} = P^{E}_{1,2} + (\delta_{1,1}k_{1,2|1} - \delta_{1,2}[k_{1,2|1} + k_{1,2|3} + k_{1,2|4}] + \delta_{1,3}k_{1,2|3} + \delta_{1,4}k_{1,2|4}) \quad (3.52)$$

$$P^{I}_{1,3} - P^{I}_{2,3} = P^{E}_{1,3} + (\delta_{1,1}k_{1,3|1} + \delta_{1,2}k_{1,3|2} - \delta_{1,3}[k_{1,3|1} + k_{1,3|2} + k_{1,3|4}] + \delta_{1,4}k_{1,3|4}) \quad (3.53)$$

$$P^{I}_{1,4} - P^{I}_{2,4} = P^{E}_{1,4} + (\delta_{1,1}k_{1,4|1} + \delta_{1,2}k_{1,4|2} + \delta_{1,3}k_{1,4|3} - \delta_{1,4}[k_{1,4|1} + k_{1,4|2} + k_{1,4|3}]) \quad (3.54)$$

From (3.51) to (3.54) the stiffness matrix for four columns is

$$\begin{bmatrix} -\Sigma k & k_{1,1|2} & k_{1,1|3} & k_{1,1|4} \\ k_{1,2|1} & -\Sigma k & k_{1,2|3} & k_{1,2|4} \\ k_{1,3|1} & k_{1,3|2} & \Sigma k & k_{1,3|4} \\ k_{1,4|1} & k_{1,4|2} & k_{1,4|3} & \Sigma k \end{bmatrix}$$

where

Σk is the sum of the each stiffness relation on the same row.

By expanding the above 4 x 4 matrix to an *m* x *m* matrix, for *m* columns at storey one, the general stiffness matrix can be written as

$$\begin{bmatrix}
-\Sigma k & k_{1,1|2} & \cdots\cdots & k_{1,1|j} & k_{1,1|j+1} & \cdots\cdots & k_{1,1|m-1} & k_{1,1|m} \\
k_{1,2|1} & -\Sigma k & \cdots\cdots & k_{1,2|j} & k_{1,2|j+1} & \cdots\cdots & k_{1,2|m-1} & k_{1,2|m} \\
\cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\
k_{1,j|1} & k_{1,j|2} & \cdots\cdots & -\Sigma k & k_{1,j|j+1} & \cdots\cdots & k_{1,j|m-1} & k_{1,j|m} \\
k_{1,j+1|1} & k_{1,j+1|2} & \cdots\cdots & k_{1,j+1|j} & -\Sigma k & \cdots\cdots & k_{1,j+1|m-1} & k_{1,j+1|m} \\
\cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\
k_{1,m-1|1} & k_{1,m-1|2} & \cdots\cdots & k_{1,m-1|j} & k_{1,m-1|j+1} & \cdots\cdots & -\Sigma k & k_{1,m-1|m} \\
k_{1,m|1} & k_{1,m|2} & \cdots\cdots & k_{1,m|j} & k_{1,m|j+1} & \cdots\cdots & k_{1,m|m-1} & -\Sigma k
\end{bmatrix}$$

General stiffness matrix 3.7 for storey one

Each and every other storey follows the same format, however the stiffness relations for each pair are those for the corresponding storey. For the general storey *i* with *m* columns per storey the stiffness matrix is

$$\begin{bmatrix}
-\Sigma k & k_{i,1|2} & \cdots\cdots & k_{i,1|j} & k_{i,1|j+1} & \cdots\cdots & k_{i,1|m-1} & k_{i,1|m} \\
k_{i,2|1} & -\Sigma k & \cdots\cdots & k_{i,2|j} & k_{i,2|j+1} & \cdots\cdots & k_{i,2|m-1} & k_{i,2|m} \\
\cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\
k_{i,j|1} & k_{i,j|2} & \cdots\cdots & -\Sigma k & k_{i,j|j+1} & \cdots\cdots & k_{i,j|m-1} & k_{i,j|m} \\
k_{i,j+1|1} & k_{i,j+1|2} & \cdots\cdots & k_{i,j+1|j} & -\Sigma k & \cdots\cdots & k_{i,j+1|m-1} & k_{i,j+1|m} \\
\cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\
k_{i,m-1|1} & k_{i,m-1|2} & \cdots\cdots & k_{i,m-1|j} & k_{i,m-1|j+1} & \cdots\cdots & -\Sigma k & k_{i,m-1|m} \\
k_{i,m|1} & k_{i,m|2} & \cdots\cdots & k_{i,m|j} & k_{i,m|j+1} & \cdots\cdots & k_{i,m|m-1} & -\Sigma k
\end{bmatrix}$$

General stiffness matrix 3.8 for any arbitrary storey i

With the matrix equation and the stiffness matrix developed for a full TCB it is possible to perform axial shortening calculations. The general matrix equation 3.6 developed assumes a fully constructed structure so analysis is not possible at this stage during the construction process. Chapter 4 considers the matrix equation as it applies to the building during the construction cycle. A detailed procedure of applying the matrix equation is developed and a worked example is given in this chapter.

3.4 Summary to Chapter

Definitions of the types of connections used in multi-storey structures and a current design approach with respect to connection details is presented in this chapter. From this, the way connections cause load sharing and the mechanics of such sharing is defined. This allows a detailed structured set of equations to be developed that link all connecting elements. In doing so, a number of matrix equations are developed that isolate the relevant axial components for an axial shortening analysis. By incorporating the long term creep and shrinkage models into the matrix equations, analysis can be made of differential axial shortening taking into account framed connections.

APPENDIX

A3.1 Reinforced concrete details

Standard concrete details for column and beam connections are available from Reinforced Concrete publications by *Steel reinforced Institute of Australia* (Woolside 1995). Structural engineers will also develop site specific connections. Below are details adopted by Placzek (1997) from Meinhardt pty ltd showing full moment connection details for columns and band beams.

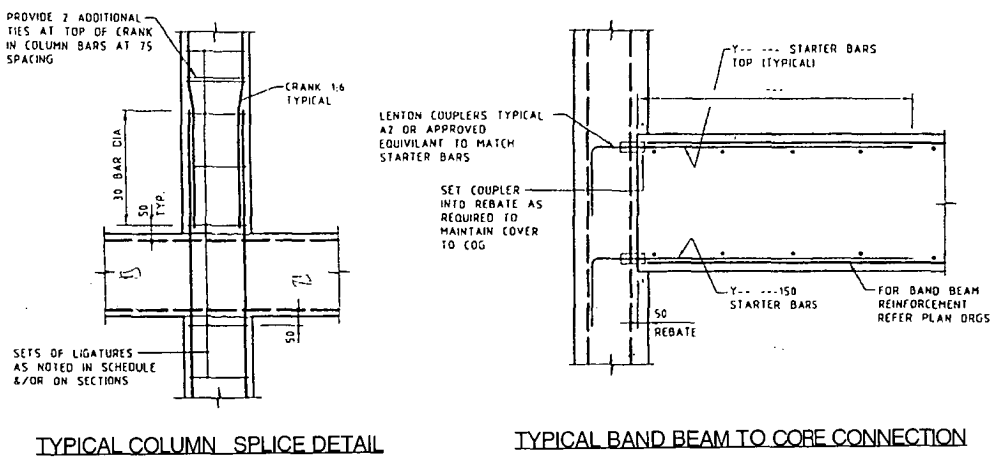


Figure A3.1 Details of reinforcing arrangement
for full moment connections Placzek (1997)

In addition to column/beam connections, plates with band beams and edge beams are used to form a stiffened floor connecting vertical elements together.

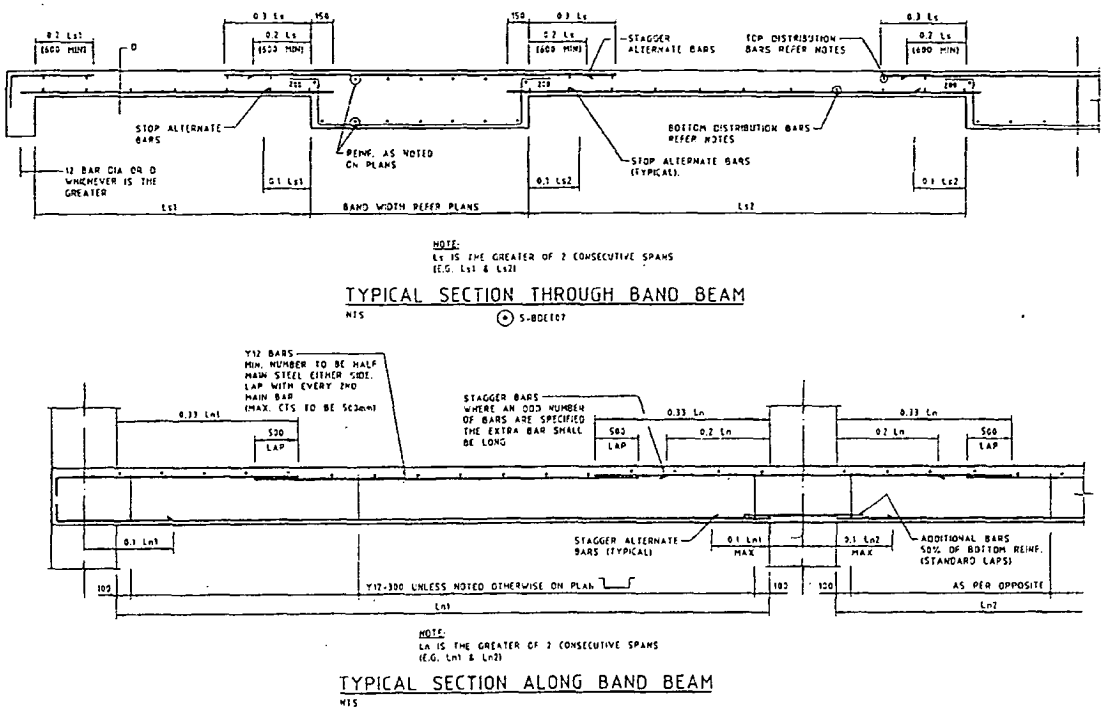


Figure A3.2 Details of reinforcing arrangement for band beams and plates Placzek (1997)

Chapter 4

APPLICATION OF FRAMING MATRICES

4.1 Introduction

In the previous chapter, matrix equations and the associated stiffness matrices were derived. The general matrix equation 3.6 is derived in the previous chapter for a complete structure and enables axial shortening to be calculated with framing between every member on a particular floor. In the case of a fully constructed building the equations will allow internal forces associated with long-term second-order effects to be calculated. However as creep and shrinkage both cause axial deformations of vertical elements from an early stage in the life of each element, it is essential that calculations can be made at various stages during the construction phase of a TCB. Each different stage involved in the building cycle needs to be considered. Matrix equation 3.6 needs to be modified accordingly to allow calculations to be made at different stages of the building cycle.

4.2 Building Cycle

As creep and shrinkage both begin from the first day of construction (Bazant 1972), it is important that the building cycle is investigated to allow a procedural model to be formulated and implemented when making axial shortening calculations.

Although most TCBs in Australia are built around a concrete core (Beasley 1987), this core is essentially just another vertical element. Around this core columns, or occasionally walls, are constructed which are then fixed to the core by beams or floor elements. A typical staged building cycle is suggested by Beasley (1987). Refer Figure 2.4. Connecting elements may be fixed with full moment connections, partial moment

connections or with pinned connections. As elements are connected, any existing axial deformations in the connected columns are locked into the frame created. Any additional applied loads will be shared within the frame and will cause further axial deformations. As more frames are created the structure evolves and internal axial loads can become shared between the entire structure.

The evolution of the structure involves a number of distinguishable stages, each of which affects how loads are shared within the entire structure. The building cycle can be isolated into three main areas that can each be divided into further stages depending on how and where they are applied. Essentially the three areas are;

i) creation of a vertical element
ii) creation of a framed connection (See Section 3.2.1)
iii) applying an external load

The creation of a vertical element may be either a core element or a wall or column element. The vertical element may be the first element of a stack of columns or an extension of an existing column.

If we consider a vertical stack of more than one column as one of the main parameters of the building, then at varying stages each column stack will be altered according to one of the three points listed above. Whenever one of these points affects the stack of columns they cause a fundamental change to the building structure. Thus each change to any stack of columns designates a stage in the building cycle. Due to the fact that individual stacks of columns are connected together by frames, then any change to one stack of columns has the potential to change the entire structure.

At every change in the structure, the axial deformations of each and every element at the time of change needs to be determined. Due to the creep and shrinkage properties of each element, the framing process is further complicated as internal loads are created by framed elements that creep and shrink. It is essential that the exact location of every point is known at every stage so that when a new column or floor element is added to

the structure we can locate the existing axial deformations of every element before a new member is added and any framing action locks[1] in existing deformations. In order to determine total axial deformations, three definitions of axial deformations are required as expressed below;

i) discrete axial deformation components
ii) local axial deformation
iii) global axial deformation.

Discrete axial deformations are creep, shrinkage, reinforcing stiffening effects or elastic deformations as defined in Section 2.3.3

Local axial deformation is the sum of the discrete axial deformations[2] of a particular element at a particular point in time.

Global axial deformation is the axial deformation of a particular element at a particular point of time measured from a fixed point, which in most cases is the base line[3]. In order to evaluate the global axial deformation for a column, the local axial deformations for each column in the column stack are summed together.

In order to appreciate how the process evolves, consider a column stack in isolation. The components of axial deformations are displayed in graph form and the associated matrix equations to calculate the generated internal forces at each stage are developed. Figure 4.1 below represents all the assumed stages of the building cycle for a TCB.

---

[1] The term *locks* refers to the way framing alters the way creep is experienced in a column before and after framing occurs. Due to load sharing after framing the internal creep loads are altered

[2] Based on the principle of superposition discussed previously

[3] In some cases the base line may not be considered the ground, however, such situations are not considered in this thesis

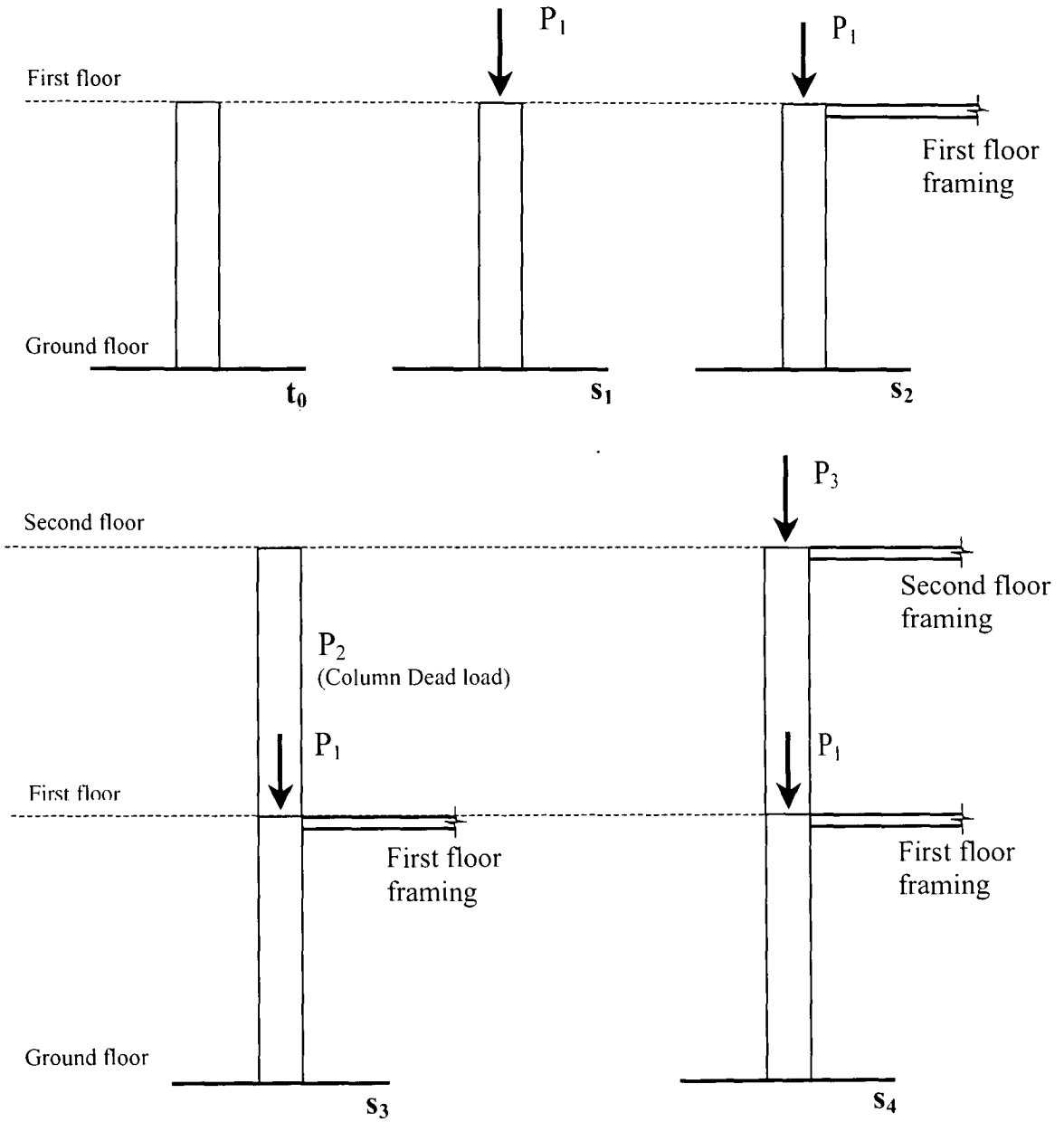


Figure 4.1 Building cycle showing at least one of the three areas of the building cycle at each stage

The five stages are displayed in Figure 4.2 showing all the discrete axial shortening components for the first storey of the column stack.

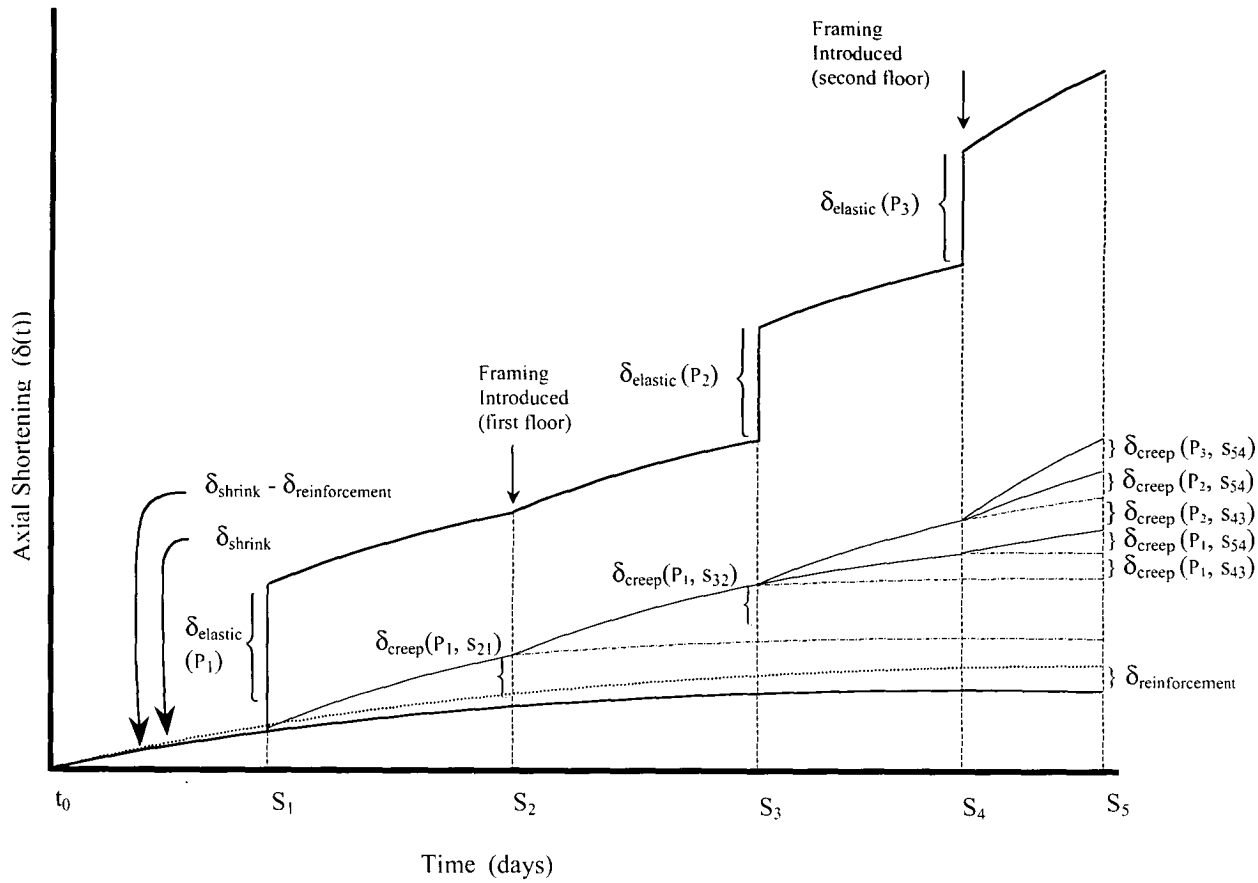


Figure 4.2 Axial shortening curves for the first storey column with applied loads and framing action as per Figure 4.1

Figure 4.2 demonstrates how the introduction of framing changes the creep curve[4]. Some points of interest are points S_2 and S_3. At point S_2 the creep curve for the axial load P_1 changes due to the first floor framing. At S_3 another axial load P_2 is applied, however the creep curve for P_1 doesn t change until the second floor framing is introduced. Although framing also alters the magnitude of elastic deformations, this cannot be easily demonstrated graphically.

The discrete axial shortening components for the second column of the column stack are displayed in Figure 4.3.

[4] It should be noted that, for ease, the changes in creep curves are show pictorially to increase the creep curve. In some cases the introduction of framing will decrease the curve

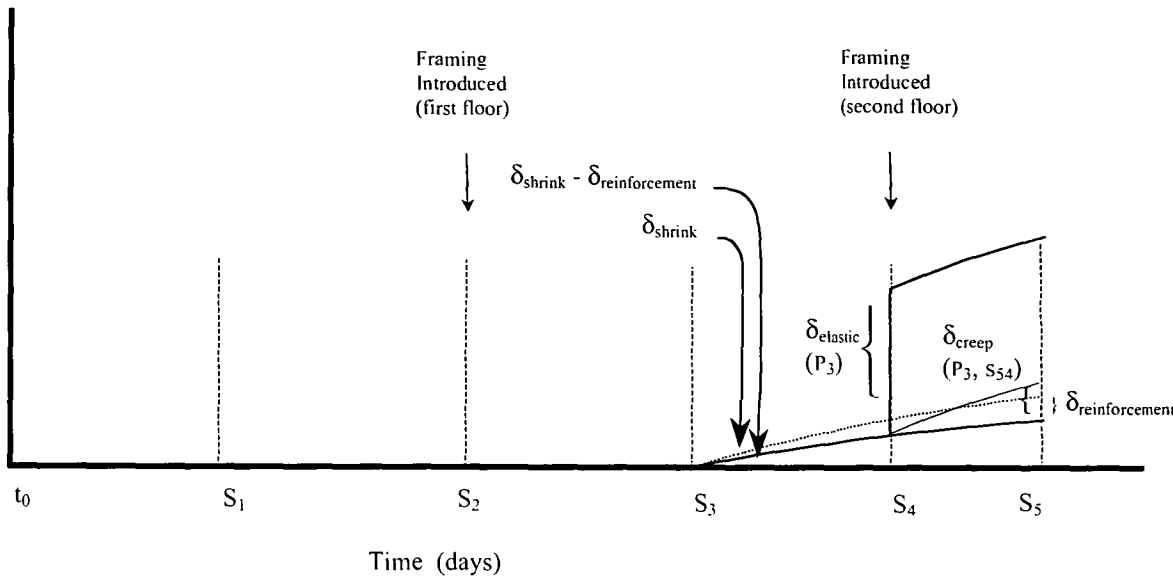


Figure 4.3 Axial shortening curves for the second storey column with applied loads and framing action as per Figure 4.1

Figure 4.2 and Figure 4.3 show the axial deformations at each point in time for each of the two columns of the column stack displayed in Figure 4.1. The local axial deformations can be found by summing each of the discrete axial deformation components at any of the stages. The global axial deformations for the first floor column in Figure 4.2 are the same as the local axial deformations as this column is the first column of the column stack. The global axial deformations for the second floor column in Figure 4.3 are found by adding the local deformations from Figure 4.3 to the local deformations for the first floor column.

At time S_3 the global axial deformations for the second floor column is greater than zero even though this is the first day of existence for the column. This is an important point as load sharing of framed elements is a factor of the differential axial deformations. Therefore it is essential that the global position of each column is known at creation so that when it is connected to another column, any pre-existing differential axial deformation is known and load sharing is only calculated from future differential axial deformations.

4.3 Calculating Axial Deformations

Axial deformation calculations are made based on an approximated building cycle for a TCB with framing occurring between columns as demonstrated by Figures 4.1 to 4.3. The assumptions made when performing these calculations are;

i) assumption of no joint rotation (refer Figure 3.4, 3.6, 3.7)
ii) stiffness relationship between columns
iii) superposition of load reversal (refer Figure 2.2)
iv) sum of large number of parts
v) iteration limitations
vi) variations in concrete properties
vii) creep and shrinkage models (refer Chapter 2.3)
viii) internal forces are assumed to be planar, axial and constant for the entire element

Due to the presence of framing, which causes load sharing to occur between column stacks, it is important that we can calculate the axial shortening at each stage. Discrete elastic axial shortening occurs instantaneously and is shared between framed members when it occurs whereas creep and shrinkage cause a progressive increase in axial shortening over time. It is not practical to distribute creep and shrinkage axial shortening due to load sharing on a day to day basis, rather creep and shrinkage axial deformations are calculated (and any load sharing made) at the end of each stage. This means that at each stage we need to make two independent calculations, one for elastic shortening and one for creep and shrinkage. To do this we need to keep track of all existing and new applied loads. This is best demonstrated by considering stages S_1 to S_3 from Figure 4.1.

At stage S_1 the applied load P_1 causes elastic axial shortening. Between S_1 and S_2 the applied load P_1 causes creep shortening. At the application of first floor framing S_2 the existing load P_1 continues to produce creep shortening, but due to load sharing, the same rate of creep between stage S_1 and S_2 cannot be used. Instead calculations are made for creep from load P_1 between stage S_2 and S_3 and distribute the creep shortening between the framed members via load sharing. Although the mechanics of the framed structure

would cause creep shortening to be shared on a day to day basis, calculations can only be made at stage S_3. Additionally at S_3 it is also necessary to calculate the elastic shortening for load P_2.

The elastic shortening needs to be made separately to the creep shortening as the load distribution due to the changing structure may be different for creep and elastic shortening. This is demonstrated at stage S_4 where a new load P_3 is added at the same time as second floor framing is introduced. Creep associated with P_1 and P_2 is calculated at S_4 and then distributed between framed members associated with the load sharing for first floor framing. However the elastic deformation due to P_3 needs to be distributed immediately between framed members associated with the load sharing for both first and second floor framing. The actual algorithms for each stage are given below.

4.3.1 Matrix equations for each stage

In all of the matrix equations below it is assumed that there are only two column stacks. Figure 4.1 depicts one of these column stacks (subscript 1). The second column stack is not shown, however distributed forces can still be calculated if we assume column stack 2 follows the same genesis as stack 1 but without any external applied loads.

Recalling the nomenclature used in the matrix equations

$P^{I}_{j,1}$ resultant internal compression force in column 1 supporting the j^{th} storey, from the equilibrium of nodal forces at the j^{th} storey

$P^{E}_{j,1}$ external force at the j^{th} storey nodes for columns 1. Such forces include all dead and live floor loads and any construction loading carried by column 1

$\delta\delta_{j,1}$ sum of the incremental time-varying axial shortening of column 1 from the first storey to the j^{th} storey

$k_{j,1|2}$ stiffness multiplier between columns 1 and 2 at the j^{th} storey.

Recalling the basic matrix equation for two columns and two storeys

$$\begin{matrix} 1^{st} \\ \text{Storey} \\ 2^{nd} \\ \text{Storey} \end{matrix} \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \atop \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right] \begin{bmatrix} P^{I}_{1,1} \\ P^{I}_{1,2} \\ \hline P^{I}_{2,1} \\ P^{I}_{2,2} \end{bmatrix} = \begin{bmatrix} P^{E}_{1,1} \\ P^{E}_{1,2} \\ \hline P^{E}_{2,1} \\ P^{E}_{2,2} \end{bmatrix} + \left[\begin{bmatrix} -k_{1,1/2} & k_{1,1/2} \\ k_{1,2/1} & -k_{1,2/1} \end{bmatrix}\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \atop \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} -k_{2,1/2} & k_{2,1/2} \\ k_{2,2/1} & -k_{2,2/1} \end{bmatrix}\right] \begin{bmatrix} \delta\delta_{1,1} \\ \delta\delta_{1,2} \\ \hline \delta\delta_{2,1} \\ \delta\delta_{2,2} \end{bmatrix}$$

Basic matrix equation 3.1

The matrix equation enables the internal forces at different stages in time to be calculated, based on the building cycle. The actual creep and elastic deformations are calculated by one of the concrete models discussed in Chapter 2. The process at arriving at the axial deformations is an iterative one as the matrix equations rely on axial deformations to find internal forces. The internal forces are then used to calculate axial deformations from the concrete models. By ensuring equilibrium of internal forces, an iterative process can be established based around the matrix equations and the concrete model. Following the development of the matrix equations below, a numerical example is given showing how the iterative process works.

The calculation of internal forces is done in two parts, one for existing loads and one for new loads. This allows creep and axial loads to be separated as discussed above.

Stage 1 $(t_0 - S_1)$

$P^{E}_{new} = 0$ $P^{E}_{old} = 0$ Framing = none

Matrix for both P^{E}_{new} and P^{E}_{old}

$$\begin{matrix} 1^{st} \\ \text{Storey} \\ 2^{nd} \\ \text{Storey} \end{matrix} \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \atop \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right] \begin{bmatrix} P^{I}_{1,1} \\ P^{I}_{1,2} \\ \hline N/A \\ N/A \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \hline 0 \\ 0 \end{bmatrix} + \left[\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \atop \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\right] \begin{bmatrix} \delta\delta_{1,1} \\ \delta\delta_{1,2} \\ \hline 0 \\ 0 \end{bmatrix}$$

As there is no external axial loading during this stage, the only axial deformations are shrinkage based.

Stage 2 (S_1 - S_2)

P^E_{new} (Start of Stage) = P_1 P^E_{old} (During Stage) = P_1 Framing = none

Matrix for P^E_{new} (the matrix equation is for calculating the elastic deformations at stage S_1)

$$\begin{matrix}1^{st}\\ \text{Storey}\\ 2^{nd}\\ \text{Storey}\end{matrix}\begin{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}\begin{bmatrix}-1&0\\0&-1\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}\end{bmatrix}\begin{bmatrix}P^I_{1,1}\\P^I_{1,2}\\\hline N/A\\N/A\end{bmatrix}=\begin{bmatrix}P_1\\0\\\hline 0\\0\end{bmatrix}+\begin{bmatrix}\begin{bmatrix}0&0\\0&0\end{bmatrix}\begin{bmatrix}0&0\\0&0\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}\begin{bmatrix}0&0\\0&0\end{bmatrix}\end{bmatrix}\begin{bmatrix}\delta\delta_{1,1}\\\delta\delta_{1,2}\\\hline 0\\0\end{bmatrix}$$

Matrix for P^E_{old} (the matrix equation is for calculating the creep deformations during the stage S_1-S_2)

$$\begin{matrix}1^{st}\\ \text{Storey}\\ 2^{nd}\\ \text{Storey}\end{matrix}\begin{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}\begin{bmatrix}-1&0\\0&-1\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}\end{bmatrix}\begin{bmatrix}P^I_{1,1}\\P^I_{1,2}\\\hline N/A\\N/A\end{bmatrix}=\begin{bmatrix}P_1\\0\\\hline 0\\0\end{bmatrix}+\begin{bmatrix}\begin{bmatrix}0&0\\0&0\end{bmatrix}\begin{bmatrix}0&0\\0&0\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}\begin{bmatrix}0&0\\0&0\end{bmatrix}\end{bmatrix}\begin{bmatrix}\delta\delta_{1,1}\\\delta\delta_{1,2}\\\hline 0\\0\end{bmatrix}$$

Stage 3 (S_2 - S_3)

P^E_{new} = 0 P^E_{old} = P_1 Framing = first floor

Matrix for P^E_{old} (the matrix equation is for calculating the creep deformations during the stage S_2-S_3)

$$\begin{matrix}1^{st}\\ \text{Storey}\\ 2^{nd}\\ \text{Storey}\end{matrix}\begin{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}\begin{bmatrix}-1&0\\0&-1\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}\end{bmatrix}\begin{bmatrix}P^I_{1.1}\\P^I_{1.2}\\\hline P^I_{2.1}\\P^I_{2.2}\end{bmatrix}=\begin{bmatrix}P_1\\0\\\hline 0\\0\end{bmatrix}+\begin{bmatrix}\begin{bmatrix}-k_{1.1/2}&k_{1.1/2}\\k_{1.2/1}&-k_{1.2/1}\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\end{bmatrix}\begin{bmatrix}\delta\delta_{1.1}\\\delta\delta_{1.2}\\\hline 0\\0\end{bmatrix}$$

Stage 4 (S_3 -S_4)

$P^E{}_{new} = P_2$ $P^E{}_{old} = P_1 + P_2$ Framing = first floor

Matrix for $P^E{}_{new}$ (the matrix equation is for calculating the elastic deformations at stage S_3

$$\begin{matrix}1^{st}\\ \text{Storey}\\ 2^{nd}\\ \text{Storey}\end{matrix}\begin{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}&\begin{bmatrix}-1&0\\0&-1\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}1&0\\0&1\end{bmatrix}\end{bmatrix}\begin{bmatrix}P^I_{1.1}\\P^I_{1.2}\\P^I_{2.1}\\P^I_{2.2}\end{bmatrix}=\begin{bmatrix}P_2\\0\\0\\0\end{bmatrix}+\begin{bmatrix}\begin{bmatrix}-k_{1.1/2}&k_{1.1/2}\\k_{1.2/1}&-k_{1.2/1}\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\end{bmatrix}\begin{bmatrix}\delta\delta_{1.1}\\\delta\delta_{1.2}\\\delta\delta_{2.1}\\\delta\delta_{2.2}\end{bmatrix}$$

Matrix for $P^E{}_{old}$ (the matrix equation is for calculating the creep deformations during the stage S_3-S_4)

$$\begin{matrix}1^{st}\\ \text{Storey}\\ 2^{nd}\\ \text{Storey}\end{matrix}\begin{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}&\begin{bmatrix}-1&0\\0&-1\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}1&0\\0&1\end{bmatrix}\end{bmatrix}\begin{bmatrix}P^I_{1.1}\\P^I_{1.2}\\P^I_{2.1}\\P^I_{2.2}\end{bmatrix}=\begin{bmatrix}P_1+P_2\\0\\0\\0\end{bmatrix}+\begin{bmatrix}\begin{bmatrix}-k_{1.1/2}&k_{1.1/2}\\k_{1.2/1}&-k_{1.2/1}\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\end{bmatrix}\begin{bmatrix}\delta\delta_{1.1}\\\delta\delta_{1.2}\\\delta\delta_{2.1}\\\delta\delta_{2.2}\end{bmatrix}$$

Stage 5 (S_4 - S_5)

$P^E{}_{new} = P_3$ $P^E{}_{old} = P_1+P_2+P_3$ Framing = first and second floors

Matrix for $P^E{}_{new}$ (the matrix equation is for calculating the elastic deformations at stage S_4

$$\begin{matrix}1^{st}\\ \text{Storey}\\ 2^{nd}\\ \text{Storey}\end{matrix}\begin{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}&\begin{bmatrix}-1&0\\0&-1\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}1&0\\0&1\end{bmatrix}\end{bmatrix}\begin{bmatrix}P^I_{1.1}\\P^I_{1.2}\\P^I_{2.1}\\P^I_{2.2}\end{bmatrix}=\begin{bmatrix}0\\0\\P_3\\0\end{bmatrix}+\begin{bmatrix}\begin{bmatrix}-k_{1.1/2}&k_{1.1/2}\\k_{1.2/1}&-k_{1.2/1}\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}-k_{2.1/2}&k_{2.1/2}\\k_{2.2/1}&-k_{2.2/1}\end{bmatrix}\end{bmatrix}\begin{bmatrix}\delta\delta_{1.1}\\\delta\delta_{1.2}\\\delta\delta_{2.1}\\\delta\delta_{2.2}\end{bmatrix}$$

Matrix for P^E_{old} (the matrix equation is for calculating the creep deformations during the stage S_4-S_5)

$$\begin{matrix}1^{st}\\ \text{Storey}\\ 2^{nd}\\ \text{Storey}\end{matrix}\left[\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}-1 & 0\\ 0 & -1\end{bmatrix}\right]\left[\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\right]\begin{bmatrix}P^I_{1.1}\\ P^I_{1.2}\\ \hline P^I_{2.1}\\ P^I_{2.2}\end{bmatrix} = \begin{bmatrix}P_1+P_2\\ 0\\ \hline P_3\\ 0\end{bmatrix} + \left[\begin{bmatrix}-k_{1.1/2} & k_{1.1/2}\\ k_{1.2/1} & -k_{1.2/1}\end{bmatrix}\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}\right]\left[\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}-k_{2.1/2} & k_{2.1/2}\\ k_{2.2/1} & -k_{2.2/1}\end{bmatrix}\right]\begin{bmatrix}\delta\delta_{1.1}\\ \delta\delta_{1.2}\\ \hline \delta\delta_{2.1}\\ \delta\delta_{2.2}\end{bmatrix}$$

4.4 Numerical Example

The numerical example is based on the building cycle in Figure 4.1. The axial deformations are calculated by the ACI(1986) method for determining creep and shrinkage and the process adopted by COLECS(1987). Both are discussed in Chapter 2.

The matrix equations in 4.3 above are used to determine internal forces for each stage. The calculation of all discrete values of axial deformation are calculated at each stage. Appendix A4.1 contains the matrix calculations and the actual shortening values. As the process is iterative and requires a number of calculations, only the first iterative pass is given in the numerical example. The final values given are as calculated by the computer program. In most cases the process converges to a solution with 99% accuracy in approximately four passes.

The process requires three steps, all performed by computer to arrive at the equilibrium between internal forces and axial deformation at each stage in time.

i) determine internal loading based on original time varying axial deformations using the matrix equations for each stage

ii) re-calculate time varying axial deformations from new internal loading from the models of ACI(1986) and COLECS(1987)

iii) check for equilibrium between time varying axial deformations at step (i) and (ii). If unequal find average[5] and perform Step (i) again

Figure 4.4 shows a flow chart of the three steps and how the process determines equilibrium between internal forces and axial deformations.

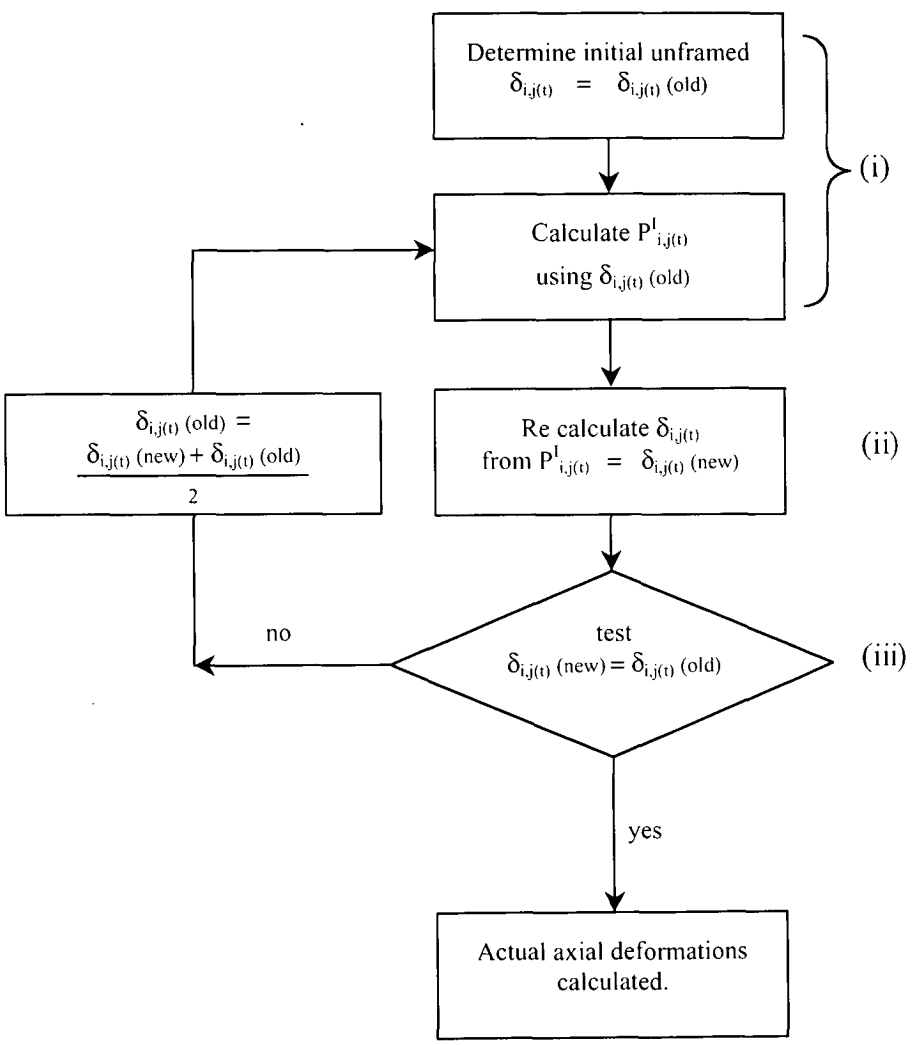


Figure 4.4 Flow chart

[5] The average is used as a basic iteration method

4.4.1 Column properties

Properties of both columns in each column stack:

Column length $_{(t=0)}$	4.0 m
28 Day strength	40 MPa
Density	2500 kg/m3
Size	500 mm x 500 mm
% reinforcing	2%

Assume basic shrinkage strain of 700 micro-strain and humidity of 50%

4.4.2 Framing properties

Floor framing (assume a spacing of 4.0 m)

First floor framing member $EI = 3.5 \times 10^5$ kN.m^2

Second floor framing member $EI = 2.8 \times 10^5$ kN.m^2

Recalling that in the stiffness matrix proposed, the framing effect is expressed in terms of the stiffness multiplier reflecting the transfer of shear forces through the framing member

$$F_{j,1|2} = k_{j,1|2}\,\Delta^{D}_{j,1|2} \qquad (4.1)$$

And for a conventional isotropic beam $k_{1,1|2}$ is simply $\frac{12E_{1|2}I_{1|2}}{\ell_{1|2}^{3}}$

Thus

$$k_{1.1|2} = \frac{12\times3.5\times10^{5}}{4^{3}} = 65.6\times10^{3}\ \text{kN/m} \qquad \text{(First floor framing)}$$

$$k_{2.1|2} = \frac{12\times2.8\times10^{5}}{4^{3}} = 52.5\times10^{3}\ \text{kN/m} \qquad \text{(Second floor framing)}$$

4.4.3 Loading

Applied external loading

P_1 = 2000 kN

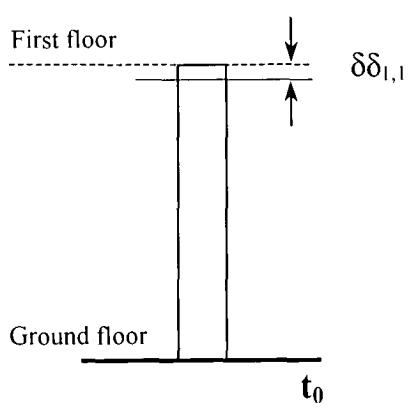
P_2 = 0.5 x 0.5 x 4 x 2500 x 9.81 / 1000 = 24.5 kN

P_3 = 1200 kN

Figures 4.1, 4.2 and 4.3 only show column stack 1. When performing load sharing calculations both column stacks are required. For simplicity we will assume that column stack 2 follows the same genesis as column stack 1 but there are no external loads applied to stack 2. Additionally we will assume that the same rigid connection is made at each floor so that the stiffness $k_{1,1|2}$ is the same for both column stacks.

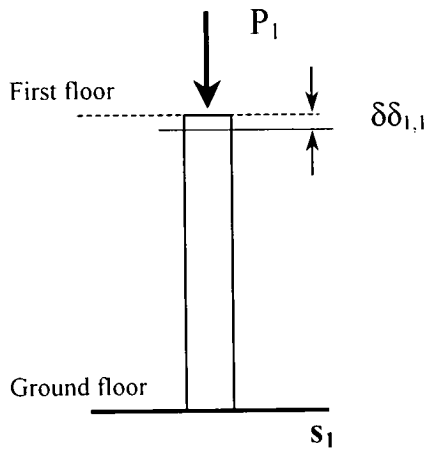
The following five stages and the relevant matrix equations as outlined above are used to determine actual shortening values for the example above. Calculations can be found in Appendix A4.2

Stage 1 (t_0 - S_1) Day 28



Axial shortening value $\delta\delta_{1,1}$ is caused by shrinkage only.

Stage 2 ($S_1 - S_2$) Day 53



Elastic axial deformation

$P^E{}_{new} = 2000kN$

Matrix equations for stage 2 axial loads (after iteration)

$$\begin{matrix}1^{st}\\ \text{Storey}\\ 2^{nd}\\ \text{Storey}\end{matrix}\begin{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}&\begin{bmatrix}-1&0\\0&-1\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}1&0\\0&1\end{bmatrix}\end{bmatrix}\begin{bmatrix}P^I_{1,1}\\P^I_{1,2}\\\text{N/A}\\\text{N/A}\end{bmatrix}=\begin{bmatrix}2000\\0\\0\\0\end{bmatrix}+\begin{bmatrix}\begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\end{bmatrix}\begin{bmatrix}1.17\\0\\0\\0\end{bmatrix}$$

Long term creep and shrinkage axial deformation

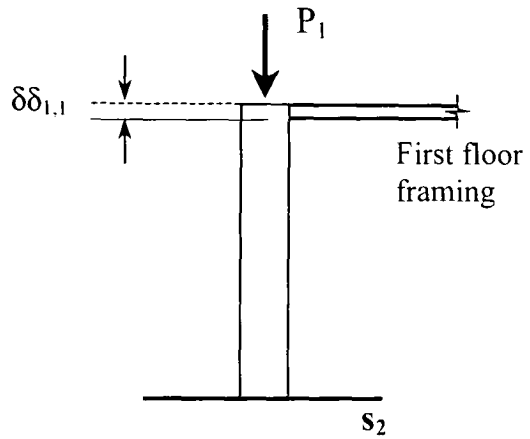
$P^E{}_{old} = 2000kN$

Matrix equations for stage 2 long term loads (after iteration)

$$\begin{matrix}1^{st}\\ \text{Storey}\\ 2^{nd}\\ \text{Storey}\end{matrix}\begin{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}&\begin{bmatrix}-1&0\\0&-1\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}1&0\\0&1\end{bmatrix}\end{bmatrix}\begin{bmatrix}P^I_{1,1}\\P^I_{1,2}\\\text{N/A}\\\text{N/A}\end{bmatrix}=\begin{bmatrix}2000\\0\\0\\0\end{bmatrix}+\begin{bmatrix}\begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\end{bmatrix}\begin{bmatrix}1.51\\0.43\\0\\0\end{bmatrix}$$

Axial shortening value $\delta\delta_{1,1}$ is the sum of both elastic and long term shortening.

Stage 3 (S_2 - S_3) Day 93



Elastic axial deformation

$P^{E}_{new} = 0$

Therefore no elastic axial shortening is produced.

Long term creep and shrinkage axial deformation

$P^{E}_{old} = 2000kN$

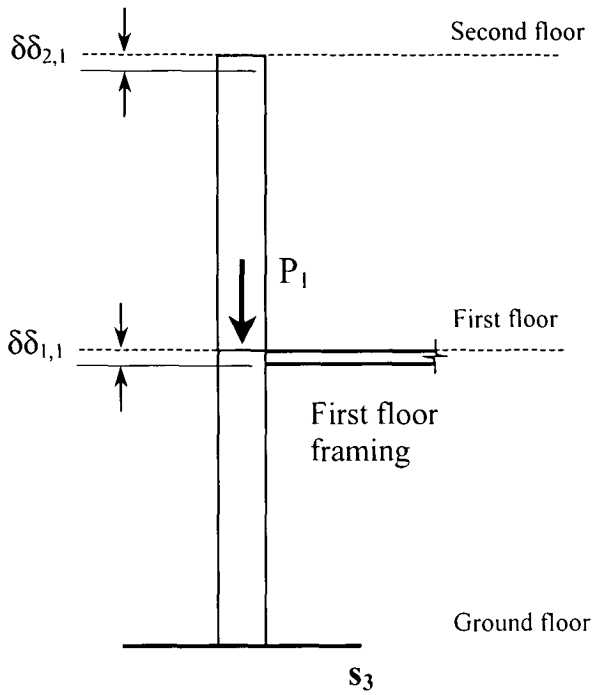
Matrix equations for stage 3 long term loads (after iteration)

$$\begin{matrix} 1^{st} \\ \text{Storey} \\ 2^{nd} \\ \text{Storey} \end{matrix} \begin{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix} \begin{bmatrix} P^{I}_{1,1} \\ P^{I}_{1,2} \\ \hline N/A \\ N/A \end{bmatrix} = \begin{bmatrix} 2000 \\ 0 \\ \hline 0 \\ 0 \end{bmatrix} + \begin{bmatrix} \begin{bmatrix} -65.6 & 65.6 \\ 65.6 & -65.6 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \end{bmatrix} \begin{bmatrix} 1.445 \\ 0.55 \\ \hline 0 \\ 0 \end{bmatrix}$$

Although at Stage S_3 no additional loading was introduced to the structure, first floor framing was created. The matrix above is required to determine internal forces that effect creep caused by load sharing associated with framing during two stages. The process for determining creep values for discrete stages in the life of an element are discussed in Appendix A4.1

Axial shortening value $\delta\delta_{1,1}$ is a result of only long term shortening.

Stage 4 (S_3 -S_4) Day 153



Elastic axial deformation

$P^E{}_{new}$ = 24.5kN (Associated dead load of second storey column)

Matrix equations for stage 4 elastic loads (after iteration)

$$\begin{matrix}1^{st}\\ \text{Storey}\\ 2^{nd}\\ \text{Storey}\end{matrix}\begin{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}&\begin{bmatrix}-1&0\\0&-1\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}1&0\\0&1\end{bmatrix}\end{bmatrix}\begin{bmatrix}P^I_{1,1}\\ P^I_{1,2}\\ \text{N/A}\\ \text{N/A}\end{bmatrix}=\begin{bmatrix}24.5\\0\\0\\0\end{bmatrix}+\begin{bmatrix}\begin{bmatrix}-65.6&65.6\\65.6&-65.6\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\end{bmatrix}\begin{bmatrix}0.0155\\0.005\\0.0155\\0.005\end{bmatrix}$$

Long term creep and shrinkage axial deformation

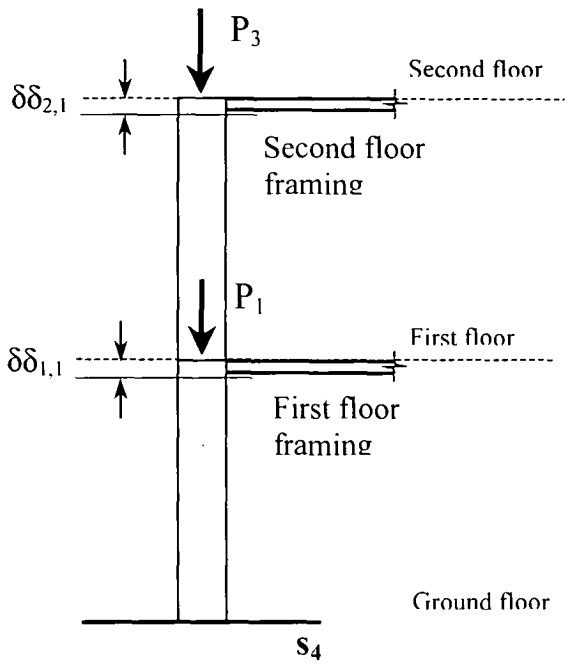
$P^E{}_{old}$ = 2024.5kN

Matrix equations for stage 4 long term loads (after iteration)

$$\begin{matrix}1^{st}\\ \text{Storey}\\ 2^{nd}\\ \text{Storey}\end{matrix}\begin{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}&\begin{bmatrix}-1&0\\0&-1\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}1&0\\0&1\end{bmatrix}\end{bmatrix}\begin{bmatrix}P^I_{1,1}\\ P^I_{1,2}\\ \text{N/A}\\ \text{N/A}\end{bmatrix}=\begin{bmatrix}2024.5\\0\\0\\0\end{bmatrix}+\begin{bmatrix}\begin{bmatrix}-65.6&65.6\\65.6&-65.6\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\end{bmatrix}\begin{bmatrix}1.14\\0.39\\1.68\\0.93\end{bmatrix}$$

Axial shortening values $\delta\delta_{1,1}$ and $\delta\delta_{2,1}$ are the sum of both elastic and long term shortening.

Stage 5 (S_4 - S_5) Day 193



Elastic axial deformation

P^{E}_{new} = 1200kN

Matrix equations for stage 5 elastic loads (after iteration)

$$\begin{matrix}1^{st}\\ \text{Storey}\\ 2^{nd}\\ \text{Storey}\end{matrix}\begin{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}\begin{bmatrix}-1&0\\0&-1\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}\end{bmatrix}\begin{bmatrix}P^{I}_{1,1}\\P^{I}_{1,2}\\\text{N/A}\\\text{N/A}\end{bmatrix}=\begin{bmatrix}0\\0\\1200\\0\end{bmatrix}+\begin{bmatrix}\begin{bmatrix}-65.6&65.6\\65.6&-65.6\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}-52.5&52.5\\52.5&-52.5\end{bmatrix}\end{bmatrix}\begin{bmatrix}0.445\\0.0151\\1.218\\0.037\end{bmatrix}$$

Long term creep and shrinkage axial deformation

P^{E}_{old} = 2024.5kN and 1200kN

Matrix equations for stage 5 long term loads (after iteration)

$$\begin{matrix}1^{st}\\ \text{Storey}\\ 2^{nd}\\ \text{Storey}\end{matrix}\begin{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}\begin{bmatrix}-1&0\\0&-1\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}\begin{bmatrix}1&0\\0&1\end{bmatrix}\end{bmatrix}\begin{bmatrix}P^{I}_{1,1}\\P^{I}_{1,2}\\\text{N/A}\\\text{N/A}\end{bmatrix}=\begin{bmatrix}2024.5\\0\\1200\\0\end{bmatrix}+\begin{bmatrix}\begin{bmatrix}-65.6&65.6\\65.6&-65.6\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\\ \begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}-52.5&52.5\\52.5&-52.5\end{bmatrix}\end{bmatrix}\begin{bmatrix}1.602\\0.794\\3.773\\1.601\end{bmatrix}$$

Axial shortening values $\delta\delta_{1,1}$ and $\delta\delta_{2,1}$ are the sum of both elastic and long term shortening.

The following curves display the axial deformations at each stage in time showing deformations for a framed and unframed column pair.

	Col $_{1,1}$		Col $_{1,2}$		Col $_{2,1}$		Col $_{2,2}$	
Stage	Unframed	Framed	Unframed	Framed	Unframed	Framed	Unframed	Framed
t_0	0	0	0	0	0	0	0	0
S_1	0.65	0.65	0.65	0.65	0	0	0	0
S_2	2.68	2.68	0.43	0.43	0	0	0	0
S_3	1.48	1.42	0.51	0.58	0	0	0	0
S_4	1.18	1.13	0.36	0.41	1.73	1.66	0.90	0.95
S_5	2.16	1.96	0.76	0.83	5.15	4.94	1.53	1.66

Figure 4.5 Total discrete axial deformations for each column at each stage from the ground. Recall nomenclature Col $_{\text{storey, column stack number}}$

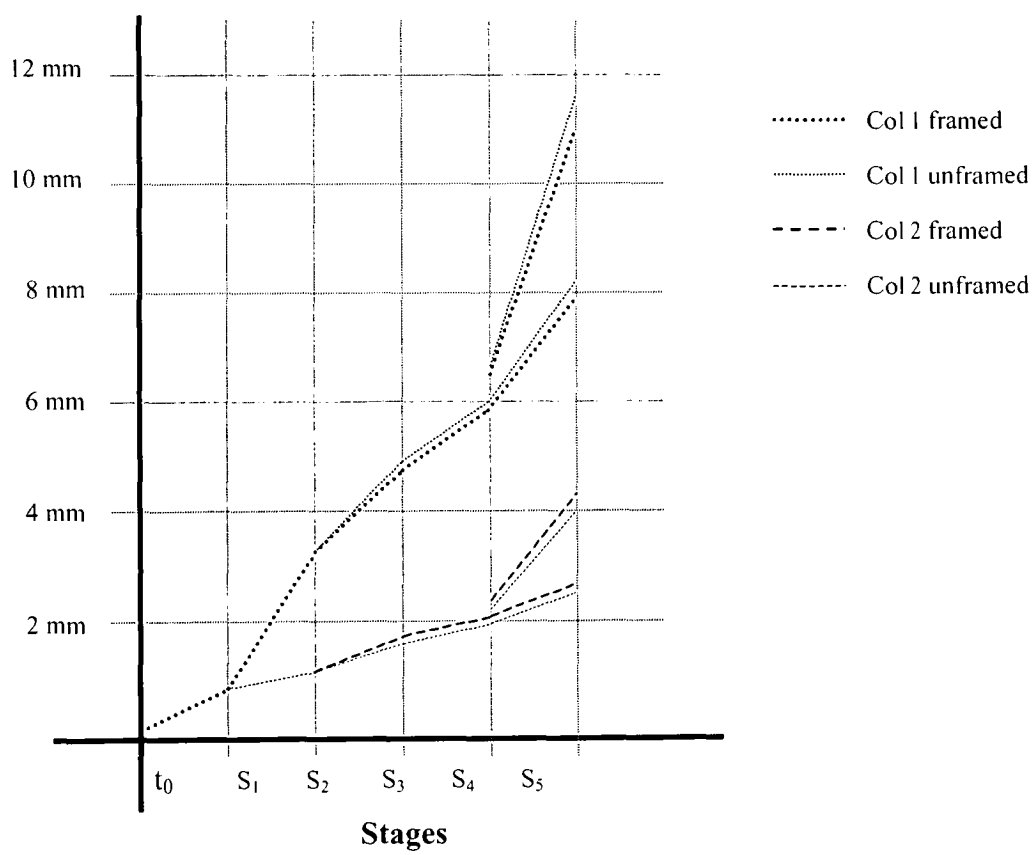


Figure 4.7 Graph of actual axial deformations at each stage

	$Col_{1,1}$		$Col_{1,2}$		$Col_{2,1}$		$Col_{2,2}$	
Stage	Unframed	Framed	Unframed	Framed	Unframed	Framed	Unframed	Framed
t_0	0	0	0	0	0	0	0	0
S_1	0.65	0.65	0.65	0.65	0	0	0	0
S_2	3.33	3.33	1.08	1.08	0	0	0	0
S_3	4.81	4.75	1.59	1.66	0	0	0	0
S_4	5.99	5.88	1.95	2.07	6.54	6.41	2.49	2.61
S_5	8.16	7.84	2.72	2.91	11.69	11.35	4.02	4.27

Figure 4.6 Total actual axial deformations for each column at each stage from the ground. Recall nomenclature $Col_{\text{storey, column stack number}}$

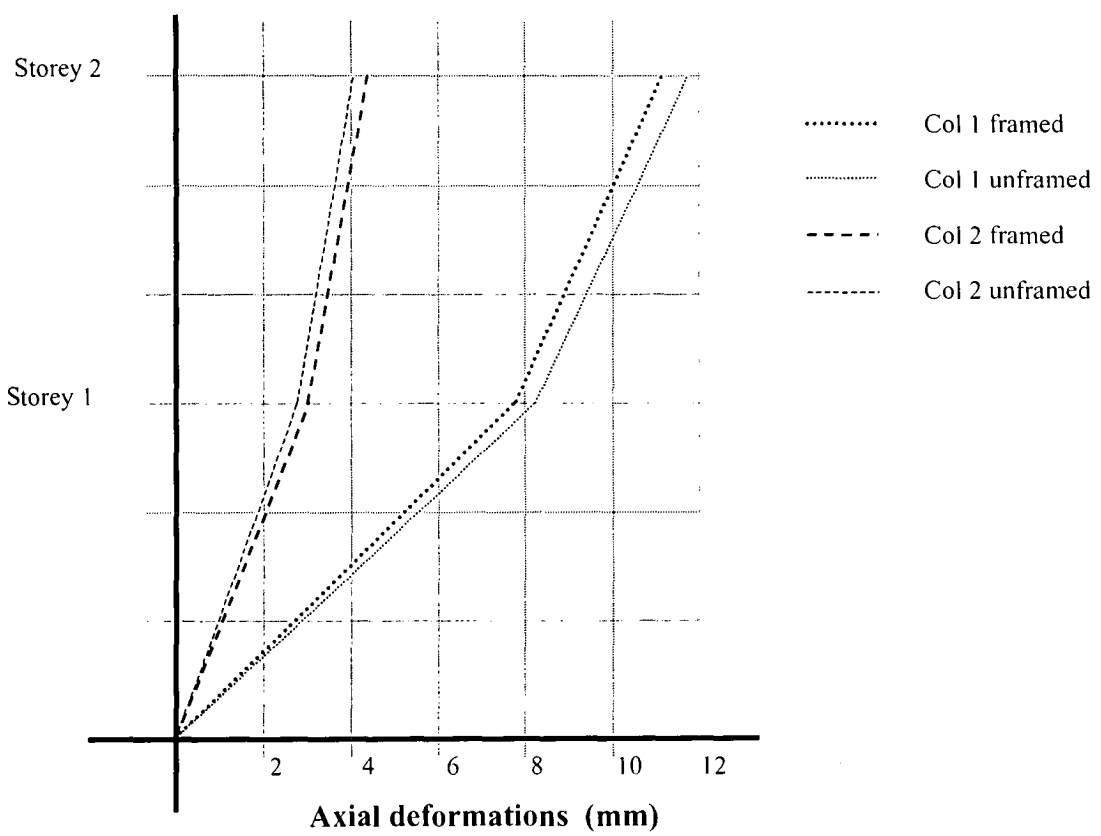


Figure 4.8 Graph showing framed vs unframed deformations at Stage S_5 for each column stack

4.5 Extended Application to Larger Structures

Calculations in the example above are based on a stiff framing member to magnify any load distribution at the early stages in the buildings life. This is done to demonstrate how the developed theory alters axial deformations through load sharing. The example above only considers two columns and two storeys, but with a stiff framing member the framing action has begun to reduce differential axial deformations by up to 10%.

The developed matrix is capable of calculating axial shortening for multiple pairs of columns, however in such examples there can be well over 100 stages, as opposed to five in the above example. Providing actual numerical examples for each stage is not appropriate given that it is essentially a repetition of the procedure above. Excel® charts of the axial shortening for two structures, one with three columns and another with four columns, are given below in Figure 4.9 and 4.10 to demonstrate the ability of the developed theory when groups of columns are analysed.

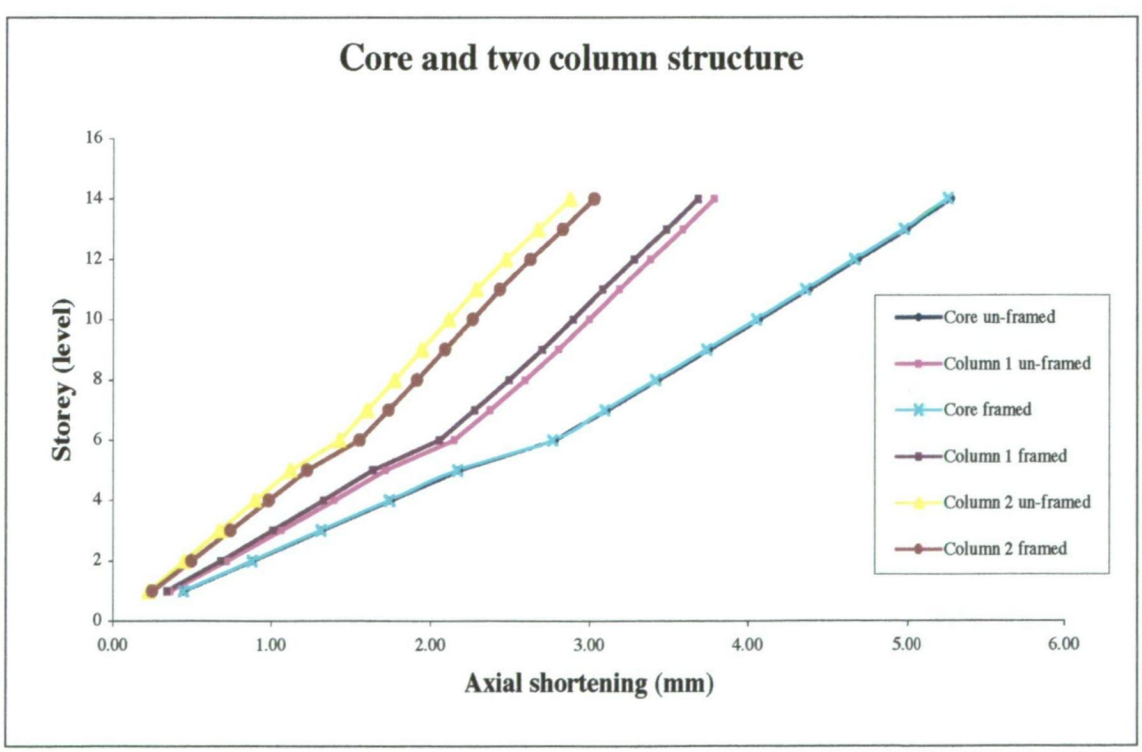


Figure 4.9 Axial shortening of 14 storey structure with core and two columns, comparing un-framed and framed analysis

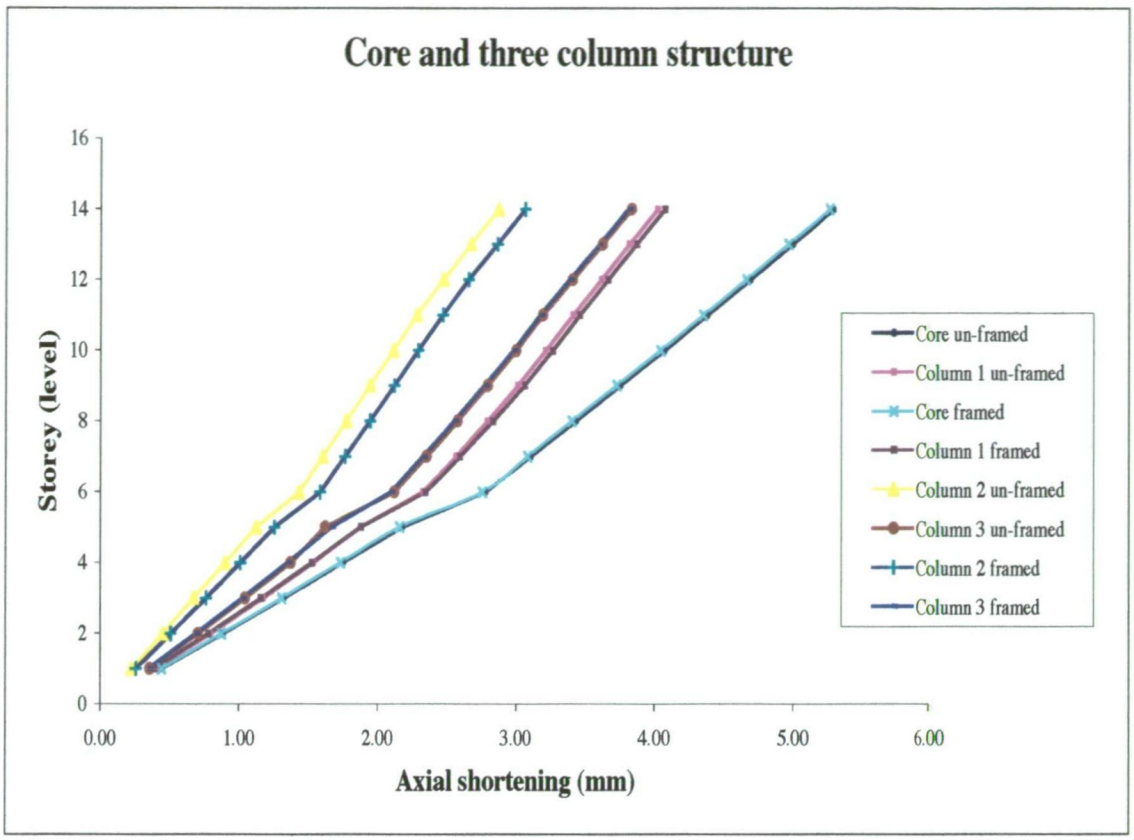


Figure 4.10 Axial shortening of 14 storey structure with core and three columns, comparing un-framed and framed analysis

Figure 4.9 and 4.10 are graphs of axial shortening values calculated by ASCA for two 14 storey buildings, showing both framed and unframed analysis. One of the structures contains three column stacks while the other contains four columns stacks. They are both more representative of a TCB than the two storey example previously analysed above as one of the three columns is modelled as a core.

The examples are given to demonstrate framing action with multiple columns. It is interesting to note that in both examples the core experiences almost no change when framing action is introduced. Additionally in both cases the column with the smallest amount of axial shortening without framing, experiences the greatest change when framing is introduced. In Chapter 5 an actual 85 storey building is modelled and the

results analysed in more detail. In this example the effect of framing becomes more apparent in the upper storeys, even in the core.

Appendix A4.3 and A4.4 contain the input data and the program results for the two 14 storey structures. The examples have been provided to demonstrate a greater application of the developed theory to framing action for multiple column groups, all interconnected with a system of beams, but are not representative of any known structure.

4.6 Summary to Chapter

The application of the matrix equation developed in Chapter 3 and how it works in conjunction with the concrete models for determining axial deformations are presented in this chapter. Of particular importance is the method for handling creep deformations with a staged building cycle. The complexity of the bookkeeping required to perform accurate analysis of axial shortening is apparent in the number of variables required for the simplest of two storey structures.

Numerical examples are given to demonstrate the iterative cycle and how the developed theory enables axial deformations to be calculated. The quantitative difference between the axial deformations which are calculated for isolated elements and framed structures is provided. To support the utility of the method developed and its application to a TCB, two examples are provided where the matrix modelling process is used to solve a system greater than a pair of columns. Both examples are outputs from the developed software package ASCA (2003).

In the authors view, this method developed, with its ability to analyse multiple column groups, demonstrates a significant step in the development of procedures for modelling and predicting differential axial shortening in framed structures.

APPENDIX

A4.1 Calculation of segmented creep axial deformations

Axial creep formula as calculated by ACI(1986), and many other methods of determining creep, provide creep coefficients at a time that is relative to the original time of construction. In most cases the total creep at a certain time can be calculated by using the relevant formula. With the age-adjusted modulus used to determine total axial creep, the resultant creep curve is usually 'S' curved. This means determining the creep for a segment between two points in time other than the start day and the final day is considerably more involved. However by using the procedure developed to determine total axial creep from day t_0 to day s_n we can find all the discrete segments of creep by considering the difference of the two time intervals required.

Consider the Figure A4.1 below showing the total creep experienced by an element between day t_0 to s_n.

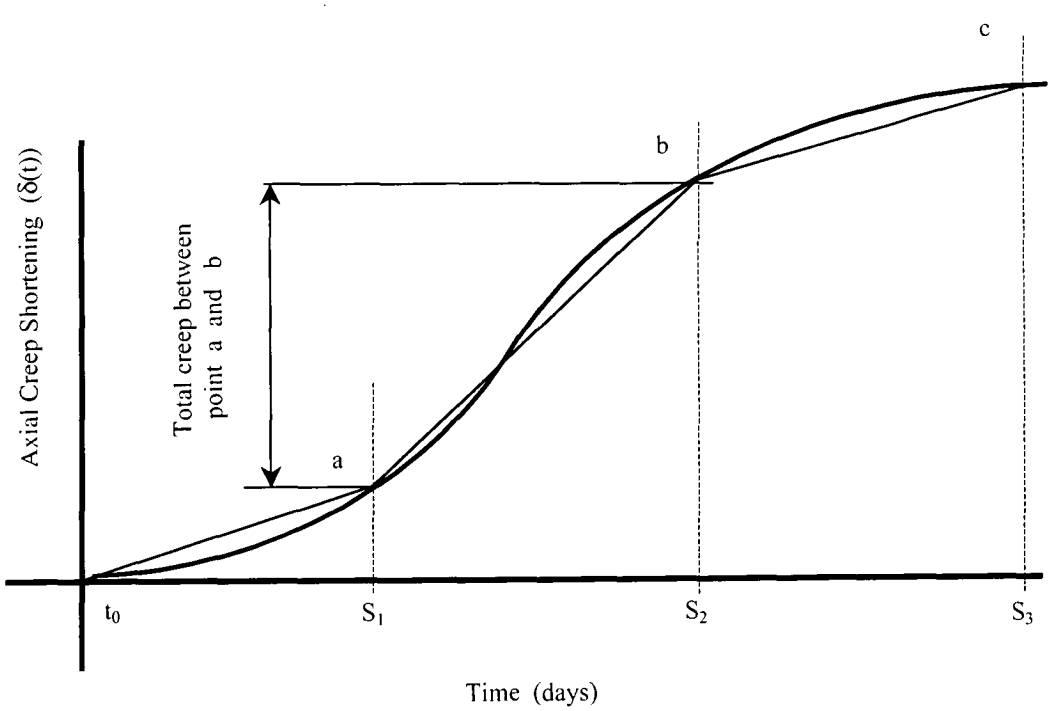


Figure A4.1 Chart of axial creep curve

By applying the ACI creep formula and the procedure developed to determine axial creep in a reinforced concrete element, the amount of actual creep at points s_1, s_2 and s_3 can all be determined. In order to determine the actual amount of creep experienced in the element between day s_1 and s_2, shown as the total creep between points *a* and *b*, the value of total creep at s_1 is subtracted from the total creep at s_2. The matrix procedure developed in Chapter 3 and Chapter 4 requires segmented values of total creep to be determined at discrete times. As it is assumed that the framing is applied at only these discrete times, the curved shape of the creep curve is not relevant, only the actual total creep experienced in these time intervals is of significance.

A4.2 Calculations of five stages of axial shortening

Stage 1 (t_0 - S_1) day 28

Step (i)

$\delta\delta_{1,1} = 0.65$ mm

$\delta\delta_{1,2} = 0.65$ mm

and

$P^I_{1,1} = 0$ kN

$\delta\delta_{1,1}$ and $\delta\delta_{1,2}$ are pure shrinkage axial deformations so no further calculation needs to be made.

Stage 2 (S_1 - S_2) day 53

Elastic axial deformation

$P^E_{new} = 2000$kN

Step (i)

$\delta\delta_{1,1} = 1.17$ mm

$\delta\delta_{1,2} = 0$ mm

and

$$\begin{matrix} 1^{st} \\ \text{Storey} \\ 2^{nd} \\ \text{Storey} \end{matrix}\left[\begin{array}{c|c} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array}\right]\left[\begin{array}{c} P^{I}_{1,1} \\ P^{I}_{1,2} \\ \hline N/A \\ N/A \end{array}\right] = \left[\begin{array}{c} 2000 \\ 0 \\ \hline 0 \\ 0 \end{array}\right] + \left[\begin{array}{c|c} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \end{array}\right]\left[\begin{array}{c} 1.17 \\ 0 \\ \hline 0 \\ 0 \end{array}\right]$$

thus

$$P^{I}_{1,1} = 2000 \text{ kN}$$
$$P^{I}_{1,2} = 0 \text{ kN}$$

Step (ii) Based on the new internal axial forces then by the same method as Step (i)

$$\delta\delta_{1,1} \text{ (new)} = 1.17 \text{ mm}$$
$$\delta\delta_{1,2} \text{ (new)} = 0 \text{ mm}$$

Step (iii) $\delta\delta_{1,1} = \delta\delta_{1,1}$ (new) and $\delta\delta_{1,2} = \delta\delta_{1,2}$ (new) [6]

thus no further iteration is required.

Long term creep and shrinkage axial deformation

$P^{E}_{old} = 2000\text{kN}$

Step (i)

$$\delta\delta_{1,1} = 1.51 \text{ mm}$$
$$\delta\delta_{1,2} = 0.43 \text{ mm}$$

and

$$\begin{matrix} 1^{st} \\ \text{Storey} \\ 2^{nd} \\ \text{Storey} \end{matrix}\left[\begin{array}{c|c} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array}\right]\left[\begin{array}{c} P^{I}_{1,1} \\ P^{I}_{1,2} \\ \hline N/A \\ N/A \end{array}\right] = \left[\begin{array}{c} 2000 \\ 0 \\ \hline 0 \\ 0 \end{array}\right] + \left[\begin{array}{c|c} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \end{array}\right]\left[\begin{array}{c} 1.51 \\ 0.43 \\ \hline 0 \\ 0 \end{array}\right]$$

[6] This could have been established simply by noting that there is no framing

thus

$P^{I}_{1,1} = 2000$ kN

$P^{I}_{1,2} = 0$ kN

again as there is no framing

$\delta\delta_{1,1} = 1.51$ mm

$\delta\delta_{1,2} = 0.43$ mm

therefore the total deformations for Stage 2 at time S_2 are

$\delta\delta_{1,1} = 1.17 + 1.51 = 2.68$ mm

$\delta\delta_{1,2} = 0 + 0.43 = 0.43$ mm

Stage 3 (S_2 - S_3) day 93

Elastic axial deformation

$P^{E}_{new} = 0$

Step (i)

$\delta\delta_{1,1} = 0$ mm

$\delta\delta_{1,2} = 0$ mm

Long term creep and shrinkage axial deformation

$P^{E}_{old} = 2000$kN

Step (i)

$\delta\delta_{1,1} = 1.48$ mm

$\delta\delta_{1,2} = 0.51$mm

$$\begin{matrix} 1^{st} \\ \text{Storey} \\ 2^{nd} \\ \text{Storey} \end{matrix} \left[\begin{array}{cc|cc} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] \left[\begin{array}{c} P^{I}_{1,1} \\ P^{I}_{1,2} \\ \hline N/A \\ N/A \end{array}\right] = \left[\begin{array}{c} 2000 \\ 0 \\ \hline 0 \\ 0 \end{array}\right] + \left[\begin{array}{cc|cc} -65.6 & 65.6 & 0 & 0 \\ 65.6 & -65.6 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right] \left[\begin{array}{c} 1.48 \\ 0.51 \\ \hline 0 \\ 0 \end{array}\right]$$

thus

$$P^{I}_{1,1} = 1936.4 \text{ kN}$$

$$P^{I}_{1,2} = 63.6 \text{ kN}$$

Step (ii) Based on the new internal axial forces then by the same method as Step (i)

$$\delta\delta_{1,1} \text{ (new)} = 1.41 \text{ mm}$$

$$\delta\delta_{1,2} \text{ (new)} = 0.59 \text{ mm}$$

Step (iii) $\delta\delta_{1,1} \neq \delta\delta_{1,1}$ (new) and $\delta\delta_{1,2} \neq \delta\delta_{1,2}$ (new)[7]

thus

$$\delta\delta_{1,1} \text{ (old)} = (1.48 + 1.41)/2 = 1.445 \text{ mm}$$

and

$$\delta\delta_{1,2} \text{ (old)} = (0.51 + 0.59)/2 = 0.55 \text{ mm}$$

Performing Step (i) with $\delta\delta_{1,1}$ (old) and $\delta\delta_{1,2}$ (old)

$$\begin{matrix} 1^{st} \\ \text{Storey} \\ 2^{nd} \\ \text{Storey} \end{matrix} \left[\begin{array}{cc|cc} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] \left[\begin{array}{c} P^{I}_{1,1} \\ P^{I}_{1,2} \\ \hline N/A \\ N/A \end{array}\right] = \left[\begin{array}{c} 2000 \\ 0 \\ \hline 0 \\ 0 \end{array}\right] + \left[\begin{array}{cc|cc} -65.6 & 65.6 & 0 & 0 \\ 65.6 & -65.6 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right] \left[\begin{array}{c} 1.445 \\ 0.55 \\ \hline 0 \\ 0 \end{array}\right]$$

[7] If only one of the deformations did not satisfy equilibrium the process would still require further iterations

therefore

$$P^{I}_{1,1} = 1941.3 \text{ kN}$$
$$P^{I}_{1,2} = 58.7 \text{ kN}$$

The following forces and axial deformations are obtained at the completion of the iteration stage.

$$P^{I}_{1,1} = 1994.9 \text{ kN}$$
$$P^{I}_{1,2} = 55.1 \text{ kN}$$

and

$$\delta\delta_{1,1} = 1.42 \text{ mm}$$
$$\delta\delta_{1,2} = 0.58 \text{ mm}$$

therefore the total deformations for Stage 3 at time S_3 are

$$\delta\delta_{1,1} = 0 + 1.42 = 1.42 \text{ mm}$$
$$\delta\delta_{1,2} = 0 + 0.58 = 0.58 \text{ mm}$$

Stage 4 (S_3 -S_4) day 133

Elastic axial deformation

$P^{E}_{new} = 24.5\text{kN}$

Step (i)

$$\delta\delta_{1,1} = 0.016 \text{ mm}$$
$$\delta\delta_{1,2} = 0 \text{ mm}$$
$$\delta\delta_{2,1} = 0.016 \text{ mm}$$
$$\delta\delta_{2,2} = 0 \text{ mm}$$

and

$$\begin{matrix} 1^{st} \\ \text{Storey} \\ 2^{nd} \\ \text{Storey} \end{matrix}\left[\begin{array}{c|c} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array}\right]\left[\begin{array}{c} P'_{1,1} \\ P'_{1,2} \\ \hline P'_{2,1} \\ P'_{2,2} \end{array}\right] = \left[\begin{array}{c} 24.5 \\ 0 \\ \hline 0 \\ 0 \end{array}\right] + \left[\begin{array}{c|c} \begin{bmatrix} -65.6 & 65.6 \\ 65.6 & -65.6 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \end{array}\right]\left[\begin{array}{c} 0.016 \\ 0 \\ \hline 0.016 \\ 0 \end{array}\right]$$

thus

$$P^{I}_{1,1} = 23.4 \text{ kN}$$

$$P^{I}_{1,2} = 1.05 \text{ kN}$$

$$P^{I}_{2,1} = 0 \text{ kN}$$

$$P^{I}_{2,2} = 0 \text{ kN}$$

Step (ii) Based on the new internal axial forces then by the same method as Step (i)

$$\delta\delta_{1,1} \text{ (new)} = 0.015 \text{ mm}$$

$$\delta\delta_{1,2} \text{ (new)} = 0.001 \text{ mm}$$

$$\delta\delta_{2,1} \text{ (new)} = 0.015 \text{ mm}$$

$$\delta\delta_{2,2} \text{ (new)} = 0.001 \text{ mm}$$

Step (iii) $\delta\delta_{1,1} \neq \delta\delta_{1,1}$ (new) and $\delta\delta_{1,2} \neq \delta\delta_{1,2}$ (new)

thus

$$\delta\delta_{1,1} \text{ (old)} = (0.016 + 0.015)/2 = 0.0155 \text{ mm}$$

and

$$\delta\delta_{1,2} \text{ (old)} = (0 + 0.001)/2 = 0.0005 \text{ mm}$$

and

$$\delta\delta_{2,1} \text{ (old)} = (0.016 + 0.015)/2 = 0.0155 \text{ mm}$$

and

$$\delta\delta_{2,2} \text{ (old)} = (0 + 0.001)/2 = 0.0005 \text{ mm}$$

Performing Step (i) with $\delta\delta_{1,1}$ (old), $\delta\delta_{1,2}$ (old), $\delta\delta_{2,1}$ (old) and $\delta\delta_{2,2}$ (old)

$$\begin{matrix} 1^{st} \\ \text{Storey} \\ 2^{nd} \\ \text{Storey} \end{matrix}\left[\begin{array}{c|c} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array}\right]\begin{bmatrix} P^{I}_{1,1} \\ P^{I}_{1,2} \\ \hline P^{I}_{2,1} \\ P^{I}_{2,2} \end{bmatrix} = \begin{bmatrix} 24.5 \\ 0 \\ \hline 0 \\ 0 \end{bmatrix} + \left[\begin{array}{c|c} \begin{bmatrix} -65.6 & 65.6 \\ 65.6 & -65.6 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \end{array}\right]\begin{bmatrix} 0.0155 \\ 0.0005 \\ \hline 0.0155 \\ 0.0005 \end{bmatrix}$$

therefore

$$\begin{aligned} P^{I}_{1,1} &= 23.45 \quad kN \\ P^{I}_{1,2} &= 1.05 \quad kN \\ P^{I}_{2,1} &= 0 \quad kN \\ P^{I}_{2,2} &= 0 \quad kN \end{aligned}$$

The following forces and axial deformations are obtained at the completion of the iteration stage.

$$\begin{aligned} P^{I}_{1,1} &= 23.45 \quad kN \\ P^{I}_{1,2} &= 1.06 \quad kN \\ P^{I}_{2,1} &= 0 \quad kN \\ P^{I}_{2,2} &= 0 \quad kN \end{aligned}$$

and

$$\begin{aligned} \delta\delta_{1,1} &= 0.0154 \quad mm \\ \delta\delta_{1,2} &= 0.001 \quad mm \\ \delta\delta_{2,1} &= 0.0154 \quad mm \\ \delta\delta_{2,2} &= 0.001 \quad mm \end{aligned}$$

Long term creep and shrinkage axial deformation

$P^{E}_{old} = 2024.5kN$

Step (i)

$$\begin{aligned} \delta\delta_{1,1} &= 1.17 \quad mm \\ \delta\delta_{1,2} &= 0.36 \quad mm \\ \delta\delta_{2,1} &= 1.71 \quad mm \\ \delta\delta_{2,2} &= 0.90 \quad mm \end{aligned}$$

and

$$\begin{matrix}1^{st}\\ \text{Storey}\\ 2^{nd}\\ \text{Storey}\end{matrix}\left[\begin{array}{c|c}\begin{bmatrix}1&0\\0&1\end{bmatrix}&\begin{bmatrix}-1&0\\0&-1\end{bmatrix}\\\hline\begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}1&0\\0&1\end{bmatrix}\end{array}\right]\left[\begin{array}{c}P'_{1,1}\\P'_{1,2}\\\hline P'_{2,1}\\P'_{2,2}\end{array}\right]=\left[\begin{array}{c}2024.5\\0\\\hline 0\\0\end{array}\right]+\left[\begin{array}{c|c}\begin{bmatrix}-65.6&65.6\\65.6&-65.6\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\\\hline\begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\end{array}\right]\left[\begin{array}{c}1.17\\0.36\\\hline 1.71\\0.90\end{array}\right]$$

thus

$$\begin{aligned}P^{I}_{1,1} &= 1971.4 \text{ kN}\\ P^{I}_{1,2} &= 53.1 \text{ kN}\\ P^{I}_{2,1} &= 0 \text{ kN}\\ P^{I}_{2,2} &= 0 \text{ kN}\end{aligned}$$

Step (ii) Based on the new internal axial forces then by the same method as Step (i)

$$\begin{aligned}\delta\delta_{1,1}\ (\text{new}) &= 1.11 \text{ mm}\\ \delta\delta_{1,2}\ (\text{new}) &= 0.42 \text{ mm}\\ \delta\delta_{2,1}\ (\text{new}) &= 1.65 \text{ mm}\\ \delta\delta_{2,2}\ (\text{new}) &= 0.96 \text{ mm}\end{aligned}$$

Step (iii) $\delta\delta_{1,1} \neq \delta\delta_{1,1}\ (\text{new})$ and $\delta\delta_{1,2} \neq \delta\delta_{1,2}\ (\text{new})$

thus

$$\delta\delta_{1,1}\ (\text{old}) = (1.17 + 1.11)/2 = 1.14 \text{ mm}$$

and

$$\delta\delta_{1,2}\ (\text{old}) = (0.36 + 0.42)/2 = 0.39 \text{ mm}$$

and

$$\delta\delta_{2,1}\ (\text{old}) = (1.71 + 1.65)/2 = 1.68 \text{ mm}$$

and

$$\delta\delta_{2,2}\ (\text{old}) = (0.90 + 0.96)/2 = 0.93 \text{ mm}$$

Performing Step (i) with $\delta\delta_{1,1}$ (old), $\delta\delta_{1,2}$ (old), $\delta\delta_{2,1}$ (old) and $\delta\delta_{2,2}$ (old)

$$\begin{matrix}1^{st}\\ \text{Storey}\\ 2^{nd}\\ \text{Storey}\end{matrix}\left[\begin{array}{c|c}\begin{bmatrix}1&0\\0&1\end{bmatrix}&\begin{bmatrix}-1&0\\0&-1\end{bmatrix}\\\hline\begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}1&0\\0&1\end{bmatrix}\end{array}\right]\left[\begin{array}{c}P'_{1,1}\\P'_{1,2}\\\hline P'_{2,1}\\P'_{2,2}\end{array}\right]=\left[\begin{array}{c}2024.5\\0\\\hline 0\\0\end{array}\right]+\left[\begin{array}{c|c}\begin{bmatrix}-65.6&65.6\\65.6&-65.6\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\\\hline\begin{bmatrix}0&0\\0&0\end{bmatrix}&\begin{bmatrix}0&0\\0&0\end{bmatrix}\end{array}\right]\left[\begin{array}{c}1.14\\0.39\\\hline 1.68\\0.93\end{array}\right]$$

therefore

$P^{I}_{1,1}$ = 1975.3 kN
$P^{I}_{1,2}$ = 49.2 kN
$P^{I}_{2,1}$ = 0 kN
$P^{I}_{2,2}$ = 0 kN

The following forces and axial deformations are obtained at the completion of the iteration stage.

$P^{I}_{1,1}$ = 1977.9 kN
$P^{I}_{1,2}$ = 46.6 kN
$P^{I}_{2,1}$ = 0 kN
$P^{I}_{2,2}$ = 0 kN

and

$\delta\delta_{1,1}$ = 1.12 mm
$\delta\delta_{1,2}$ = 0.41 mm
$\delta\delta_{2,1}$ = 1.66 mm
$\delta\delta_{2,2}$ = 0.95 mm

therefore the total deformations for Stage 4 at time S_4 are

$\delta\delta_{1,1}$ = 0.0154 + 1.12 = 1.1354 mm
$\delta\delta_{1,2}$ = 0.001 + 0.41 = 0.411 mm
$\delta\delta_{2,1}$ = 0.0154 + 1.66 = 1.6654 mm
$\delta\delta_{2,2}$ = 0.001 + 0.95 = 0.951 mm

Stage 5 (S_4 - S_5) day 193

Elastic axial deformation

P^{E}_{new} = 1200kN

Step (i)

$\delta\delta_{1,1}$ = 0.48 mm
$\delta\delta_{1,2}$ = 0 mm
$\delta\delta_{2,1}$ = 1.29 mm
$\delta\delta_{2,2}$ = 0 mm

and

$$\begin{matrix} 1^{st} \\ \text{Storey} \\ 2^{nd} \\ \text{Storey} \end{matrix}\left[\begin{array}{c|c} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array}\right]\left[\begin{array}{c} P'_{1,1} \\ P'_{1,2} \\ \hline P'_{2,1} \\ P'_{2,2} \end{array}\right] = \left[\begin{array}{c} 0 \\ 0 \\ \hline 1200 \\ 0 \end{array}\right] + \left[\begin{array}{c|c} \begin{bmatrix} -65.6 & 65.6 \\ 65.6 & -65.6 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} -52.5 & 52.5 \\ 52.5 & -52.5 \end{bmatrix} \end{array}\right]\left[\begin{array}{c} 0.48 \\ 0 \\ \hline 1.29 \\ 0 \end{array}\right]$$

thus

$$P^{I}_{1,1} = 1132.2 \text{ kN}$$
$$P^{I}_{1,2} = 67.72 \text{ kN}$$
$$P^{I}_{2,1} = 1132.2 \text{ kN}$$
$$P^{I}_{2,2} = 67.72 \text{ kN}$$

Step (ii) Based on the new internal axial forces then by the same method as Step (i)

$$\delta\delta_{1,1} \text{ (new)} = 0.41 \text{ mm}$$
$$\delta\delta_{1,2} \text{ (new)} = 0.031 \text{ mm}$$
$$\delta\delta_{2,1} \text{ (new)} = 1.145 \text{ mm}$$
$$\delta\delta_{2,2} \text{ (new)} = 0.074 \text{ mm}$$

Step (iii) $\delta\delta_{1,1} \neq \delta\delta_{1,1}$ (new) and $\delta\delta_{1,2} \neq \delta\delta_{1,2}$ (new)

thus

$$\delta\delta_{1,1} \text{ (old)} = (0.48 + 0.41) / 2 = 0.445 \text{ mm}$$

and

$$\delta\delta_{1,2} \text{ (old)} = (0 + 0.031) / 2 = 0.0151 \text{ mm}$$

and

$$\delta\delta_{2,1} \text{ (old)} = (1.29 + 1.145) / 2 = 1.218 \text{ mm}$$

and

$$\delta\delta_{2,2} \text{ (old)} = (0 + 0.074) / 2 = 0.037 \text{ mm}$$

Performing Step (i) with $\delta\delta_{1,1}$ (old) , $\delta\delta_{1,2}$ (old) , $\delta\delta_{2,1}$ (old) and $\delta\delta_{2,2}$ (old)

$$\begin{matrix} 1^{st} \\ \text{Storey} \\ 2^{nd} \\ \text{Storey} \end{matrix}\left[\begin{array}{c|c} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array}\right]\left[\begin{array}{c} P'_{1,1} \\ P'_{1,2} \\ \hline P'_{2,1} \\ P'_{2,2} \end{array}\right] = \left[\begin{array}{c} 0 \\ 0 \\ \hline 1200 \\ 0 \end{array}\right] + \left[\begin{array}{c|c} \begin{bmatrix} -65.6 & 65.6 \\ 65.6 & -65.6 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} -52.5 & 52.5 \\ 52.5 & -52.5 \end{bmatrix} \end{array}\right]\left[\begin{array}{c} 0.445 \\ 0.0151 \\ \hline 1.218 \\ 0.037 \end{array}\right]$$

therefore

$$P^{I}_{1,1} = 1134.1 \text{ kN}$$
$$P^{I}_{1,2} = 65.9 \text{ kN}$$
$$P^{I}_{2,1} = 1134.1 \text{ kN}$$
$$P^{I}_{2,2} = 65.9 \text{ kN}$$

The following forces and axial deformations are obtained at the completion of the iteration stage.

$$P^{I}_{1,1} = 1135.7 \text{ kN}$$
$$P^{I}_{1,2} = 64.3 \text{ kN}$$
$$P^{I}_{2,1} = 1135.7 \text{ kN}$$
$$P^{I}_{2,2} = 64.3 \text{ kN}$$

and

$$\delta\delta_{1,1} = 0.42 \text{ mm}$$
$$\delta\delta_{1,2} = 0.032 \text{ mm}$$
$$\delta\delta_{2,1} = 1.149 \text{ mm}$$
$$\delta\delta_{2,2} = 0.075 \text{ mm}$$

Long term creep and shrinkage axial deformation

P^{E}_{old} = 2024.5kN and 1200kN

Step (i)

$$\delta\delta_{1,1} = 1.682 \text{ mm}$$
$$\delta\delta_{1,2} = 0.766 \text{ mm}$$
$$\delta\delta_{2,1} = 3.895 \text{ mm}$$
$$\delta\delta_{2,2} = 1.532 \text{ mm}$$

and

$$\begin{matrix} 1^{st} \\ \text{Storey} \\ 2^{nd} \\ \text{Storey} \end{matrix} \left[\begin{array}{c|c} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array}\right] \left[\begin{array}{c} P^{I}_{1,1} \\ P^{I}_{1,2} \\ \hline P^{I}_{2,1} \\ P^{I}_{2,2} \end{array}\right] = \left[\begin{array}{c} 2024.5 \\ 0 \\ \hline 1200 \\ 0 \end{array}\right] + \left[\begin{array}{c|c} \begin{bmatrix} -65.6 & 65.6 \\ 65.6 & -65.6 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} -52.5 & 52.5 \\ 52.5 & -52.5 \end{bmatrix} \end{array}\right] \left[\begin{array}{c} 1.682 \\ 0.766 \\ \hline 3.895 \\ 1.532 \end{array}\right]$$

thus

$$P^{I}_{1,1} = 3040.3 \text{ kN}$$

$$P^{I}_{1,2} = 184.2 \text{ kN}$$

$$P^{I}_{2,1} = 1075.9 \text{ kN}$$

$$P^{I}_{2,2} = 124.1 \text{ kN}$$

Step (ii) Based on the new internal axial forces then by the same method as Step (i)

$$\delta\delta_{1,1} \text{ (new)} = 1.522 \text{ mm}$$

$$\delta\delta_{1,2} \text{ (new)} = 0.821 \text{ mm}$$

$$\delta\delta_{2,1} \text{ (new)} = 3.650 \text{ mm}$$

$$\delta\delta_{2,2} \text{ (new)} = 1.670 \text{ mm}$$

Step (iii) $\delta\delta_{1,1} \neq \delta\delta_{1,1}$ (new) and $\delta\delta_{1,2} \neq \delta\delta_{1,2}$ (new)

thus

$$\delta\delta_{1,1} \text{ (old)} = (1.682 + 1.522) / 2 = 1.602 \text{ mm}$$

and

$$\delta\delta_{1,2} \text{ (old)} = (0.766 + 0.821) / 2 = 0.794 \text{ mm}$$

and

$$\delta\delta_{2,1} \text{ (old)} = (3.895 + 3.65) / 2 = 3.773 \text{ mm}$$

and

$$\delta\delta_{2,2} \text{ (old)} = (1.532 + 1.67) / 2 = 1.601 \text{ mm}$$

Performing Step (i) with $\delta\delta_{1,1}$ (old) , $\delta\delta_{1,2}$ (old) , $\delta\delta_{2,1}$ (old) and $\delta\delta_{2,2}$ (old

$$\begin{matrix} 1^{st} \\ \text{Storey} \\ 2^{nd} \\ \text{Storey} \end{matrix} \left[\begin{array}{c|c} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array}\right] \left[\begin{array}{c} P'_{1,1} \\ P'_{1,2} \\ \hline P'_{2,1} \\ P'_{2,2} \end{array}\right] = \left[\begin{array}{c} 2024.5 \\ 0 \\ \hline 1200 \\ 0 \end{array}\right] + \left[\begin{array}{c|c} \begin{bmatrix} -65.6 & 65.6 \\ 65.6 & -65.6 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \hline \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} -52.5 & 52.5 \\ 52.5 & -52.5 \end{bmatrix} \end{array}\right] \left[\begin{array}{c} 1.602 \\ 0.794 \\ \hline 3.773 \\ 1.601 \end{array}\right]$$

therefore

$$P^{I}_{1,1} = 3056.8 \quad kN$$
$$P^{I}_{1,2} = 167.7 \quad kN$$
$$P^{I}_{2,1} = 1085.3 \quad kN$$
$$P^{I}_{2,2} = 114.7 \quad kN$$

The following forces and axial deformations are obtained at the completion of the iteration stage.

$$P^{I}_{1,1} = 3055.7 \quad kN$$
$$P^{I}_{1,2} = 168.8 \quad kN$$
$$P^{I}_{2,1} = 1082.9 \quad kN$$
$$P^{I}_{2,2} = 117.1 \quad kN$$

and

$$\delta\delta_{1,1} \text{ (new)} = 1.541 \text{ mm}$$
$$\delta\delta_{1,2} \text{ (new)} = 0.803 \text{ mm}$$
$$\delta\delta_{2,1} \text{ (new)} = 3.790 \text{ mm}$$
$$\delta\delta_{2,2} \text{ (new)} = 1.588 \text{ mm}$$

therefore the total deformations for Stage 5 at time S_5 are

$$\delta\delta_{1,1} = 0.42 + 1.541 = 1.961 \text{ mm}$$
$$\delta\delta_{1,2} = 0.032 + 0.803 = 0.835 \text{ mm}$$
$$\delta\delta_{2,1} = 1.149 + 3.790 = 4.939 \text{ mm}$$
$$\delta\delta_{2,2} = 0.075 + 1.588 = 1.663 \text{ mm}$$

A4.3 Input data for a 14 storey structure

Heading meanings

STRY NAME	Storey name	STRY HGHT	Storey level
OCPY TIME	Day of occupancy	OCPY LOAD	Occupancy loading
CONS TIME	Days to construct storey	CONS LOAD	Construction loads applied
CONC GRDE	Concrete grade	CONC DENS	Concrete density
W/C RAT	Water Cement ratio	CMNT CONT	Cement Content
AREA /DPH	Total area or depth of section. Total area is used	PRMT /WDH	Total perimeter or width of section. Total perimeter is used
NO. BRS	Number of bars or percentage or reinforcing Percentage of reinforcing is used	BASC SHRK	Basic shrinkage strain adopted
AVG HUM	Average humidity		

The first row of each storey is the core data, and the subsequent lines are for the columns. Core and two column input data.

STRY NAME	STRY HGHT M	OCPY TIME DAYS	OCPY LOAD KN	CONS TIME DAYS	CONS LOAD KN	CONC GRDE MPA	CONC DENS KG/M3	W/C RAT	CMNT CONT KG/M3	AREA /DPH M2/M	PRMT /WDH M/M	NO. BRS /%R	BASC SHRK COEF	AVG HUM %
LG	4.00	200.	1091.	23.	3194.	50.	2400.	0.45	400.	59.84	150.00	-3.4	500.	70.
			150.	30.	439.	50.	2400.	0.45	400.	1.67	8.00	-30.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-30.0	500.	70.
1	4.00	200.	1091.	23.	1935.	50.	2400.	0.45	400.	59.84	150.00	-3.3	500.	70.
			150.	30.	266.	50.	2400.	0.45	400.	1.66	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
2	4.00	200.	1091.	23.	1935.	50.	2400.	0.45	400.	59.84	150.00	-3.0	500.	70.
			150.	30.	266.	50.	2400.	0.45	400.	1.66	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
3	4.00	200.	1091.	23.	1935.	50.	2400.	0.45	400.	59.84	150.00	-3.0	500.	70.
			150.	30.	266.	50.	2400.	0.45	400.	1.66	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
4	4.00	200.	1091.	23.	1935.	50.	2400.	0.45	400.	59.84	150.00	-2.9	500.	70.
			150.	30.	266.	50.	2400.	0.45	400.	1.66	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
5	5.55	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.6	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.66	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
6	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.6	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
7	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.5	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
8	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.3	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
9	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.3	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.

10	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.3	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
11	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.1	500.	70.
			60.	30.	316.	60.	2400.	0.45	400.	1.70	8.00	-27.0	500.	70.
			50	30	50	60.	2400.	0.45	400.	1.70	8.00	-27.0	500.	70.
12	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.1	500.	70.
			60.	14.	316.	60.	2400.	0.45	400.	1.70	8.00	-25.0	500.	70.
			50	30	50	60.	2400.	0.45	400.	1.70	8.00	-25.0	500.	70.
13	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.1	500.	70.
			60.	14.	316.	60.	2400.	0.45	400.	1.70	8.00	-25.0	500.	70.
			50	30	50	60.	2400.	0.45	400.	1.70	8.00	-25.0	500.	70.

Core and three column input data

STRY NAME	STRY HGHT M	OCPY TIME DAYS	OCPY LOAD KN	CONS TIME DAYS	CONS LOAD KN	CONC GRDE MPA	CONC DENS KG/M3	W/C RAT	CMNT CONT KG/M3	AREA /DPH M2/M	PRMT /WDH M/M	NO. BRS /%R	BASC SHRK COEF	AVG HUM %
LG	4.00	200.	1091.	23.	3194.	50.	2400.	0.45	400.	59.84	150.00	-3.4	500.	70.
			150.	30.	439.	50.	2400.	0.45	400.	1.67	8.00	-30.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-30.0	500.	70.
			50	30	250	50.	2400.	0.45	400.	1.67	8.00	-30.0	500.	70.
1	4.00	200.	1091.	23.	1935.	50.	2400.	0.45	400.	59.84	150.00	-3.3	500.	70.
			150.	30.	266.	50.	2400.	0.45	400.	1.66	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	250	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
2	4.00	200.	1091.	23.	1935.	50.	2400.	0.45	400.	59.84	150.00	-3.0	500.	70.
			150.	30.	266.	50.	2400.	0.45	400.	1.66	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	250	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
3	4.00	200.	1091.	23.	1935.	50.	2400.	0.45	400.	59.84	150.00	-3.0	500.	70.
			150.	30.	266.	50.	2400.	0.45	400.	1.66	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	250	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
4	4.00	200.	1091.	23.	1935.	50.	2400.	0.45	400.	59.84	150.00	-2.9	500.	70.
			150.	30.	266.	50.	2400.	0.45	400.	1.66	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	250	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
5	5.55	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.6	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.66	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	250	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
6	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.6	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	250	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
7	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.5	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	250	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
8	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.3	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	250	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
9	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.3	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	250	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
10	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.3	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	50	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
			50	30	250	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
11	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.1	500.	70.
			60.	30.	316.	60.	2400.	0.45	400.	1.70	8.00	-27.0	500.	70.
			50	30	50	60.	2400.	0.45	400.	1.70	8.00	-27.0	500.	70.
			50	30	250	50.	2400.	0.45	400.	1.70	8.00	-27.0	500.	70.
12	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.1	500.	70.
			60.	14.	316.	60.	2400.	0.45	400.	1.70	8.00	-25.0	500.	70.
			50	30	50	60.	2400.	0.45	400.	1.70	8.00	-25.0	500.	70.
			50	30	250	50.	2400.	0.45	400.	1.70	8.00	-25.0	500.	70.
13	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.1	500.	70.
			60.	14.	316.	60.	2400.	0.45	400.	1.70	8.00	-25.0	500.	70.
			50	30	50	60.	2400.	0.45	400.	1.70	8.00	-25.0	500.	70.
			50	30	250	50.	2400.	0.45	400.	1.70	8.00	-25.0	500.	70.

A4.4 ASCA program output results

Core and two columns

	Column and core without framing action			Column and core without framing action		
Storey	Core un-framed	Column 1 un-framed	Column 2 un-framed	Core framed	Column 1 framed	Column 2 framed
	mm	mm	mm	mm	mm	mm
1	0.45	0.36	0.23	0.44	0.34	0.25
2	0.88	0.72	0.45	0.88	0.68	0.50
3	1.32	1.06	0.68	1.31	1.01	0.74
4	1.75	1.40	0.90	1.74	1.33	0.99
5	2.18	1.72	1.12	2.17	1.64	1.23
6	2.78	2.14	1.43	2.77	2.05	1.55
7	3.11	2.37	1.60	3.09	2.27	1.74
8	3.43	2.59	1.77	3.41	2.49	1.91
9	3.75	2.80	1.94	3.74	2.70	2.09
10	4.07	3.00	2.11	4.05	2.89	2.26
11	4.38	3.19	2.28	4.36	3.08	2.43
12	4.69	3.39	2.47	4.67	3.28	2.63
13	5.00	3.59	2.67	4.98	3.48	2.83
14	5.29	3.79	2.88	5.27	3.68	3.03

Core and three columns

	Column and core without framing action				Column and core <u>with</u> framing action			
Storey	Core un-framed	Column 1 un-framed	Column 2 un-framed	Column 3 un-framed	Core framed	Column 1 framed	Column 2 framed	Column 3 framed
	mm	mm	mm	mm	mm	mm	mm	mm
1	0.45	0.40	0.23	0.36	0.44	0.39	0.26	0.35
2	0.88	0.79	0.45	0.70	0.87	0.78	0.51	0.69
3	1.32	1.17	0.68	1.04	1.31	1.16	0.76	1.02
4	1.75	1.53	0.90	1.37	1.73	1.52	1.01	1.35
5	2.18	1.88	1.12	1.62	2.16	1.88	1.25	1.66
6	2.78	2.33	1.43	2.12	2.76	2.35	1.58	2.09
7	3.11	2.57	1.60	2.35	3.09	2.60	1.76	2.33
8	3.43	2.80	1.77	2.58	3.41	2.84	1.94	2.55
9	3.75	3.01	1.94	2.80	3.73	3.06	2.12	2.77
10	4.07	3.22	2.11	3.00	4.04	3.27	2.29	2.98
11	4.38	3.41	2.28	3.20	4.35	3.46	2.47	3.18
12	4.69	3.62	2.47	3.41	4.66	3.67	2.66	3.39
13	5.00	3.82	2.67	3.63	4.97	3.87	2.86	3.61
14	5.29	4.02	2.88	3.84	5.26	4.08	3.07	3.81

Chapter 5

RESULTS OF ASCA AXIAL SHORTENING ANALYSIS

5.1 Introduction

A computer program, ASCA (2003), has been written to assist with the calculation of axial shortening results for a multi-storey building. The development of the program ASCA is outlined in Chapter 6 and Section 2 of this thesis. The program models the building during the construction cycle and takes into account the framing action of each storey in the structure. Results for two differing sets of data from an actual building are used to demonstrate framing effects on axial shortening. The data sets differ by the degree of differential axial shortening and by the shape of the axial shortening curves for the column and the core.

Validating the results is as equally important as the results themselves and whilst there are few, if any, complete sets of data of structures with axial shortening data accounting for framing action, other sets of data of completed multi-storey buildings are available for comparison.

This chapter analyses the results of these two differing sets of data then attempts to validate the results by comparison with another similar program. Finally the predicted effect of framing on differential axial shortening is objectively discussed.

5.2 Accuracy

Methods for catching and checking errors within the program are discussed further in Chapter 6. Of more importance here are the calibration and validation of the results. Although calibrating the results of a program is essential to validating results, as the program produces unique results, calibration is difficult. A number of sub-routines, namely

calculations of creep and shrinkage coefficients, the age-adjusted effective modulus and the concrete column models represent separate sources of error. However each of these sub-routines have been extensively tested as individual components of the program and calibrated against recognised data.

The overall set of results for an actual building is of critical concern. From the outset of the development of the underlying theory and the generation of the computer code, emphasis has been placed on introducing framing action to methods for calculation of axial shortening that have been successfully tested against isolated columns. The method for analysis has been based on a tried and tested program, COLECS (1987), with two actual data files of a constructed building[1]. Each data set are of differential axial shortening results for one column and the core from the Bayoke Tower II building both produced by COLECS. One set has a small differential axial shortening between the two elements whilst the other has far more significant differential shortening. In addition the differential axial shortening in Data Set 1 does not increase at each floor where as with Data Set 2 it does. Because of this the effects framing has on each data set differs considerably. As the final results for a framed structure are actually an extension of the results produced from COLECS, by manipulating the inputs into the program with framing action, two sets of results can be produced, one with framing action and one without.

Manipulating ASCA to produce results without framing action is done by switching off the stiffness matrix for framing action, and the iteration effect associated with this matrix, as the program produces results. In effect a set of non-framing results are obtained. All the processes and steps required to produce results are still followed except framing alterations are not considered. These results can then be calibrated against results from COLECS for an actual data file without framing action. Thus with framing action included, the only variation to those results are framing effects calculated by the program.

---

[1] Two data files for the Bayoke II Tower in Bangkok are used to make comparisons between the two programs

The program algorithm for modifying axial shortening in order to account for framing is actually very efficient and can be found in Appendix A5.1. Figure 5.1 below illustrates the flow chart showing the specific workings of the algorithm.

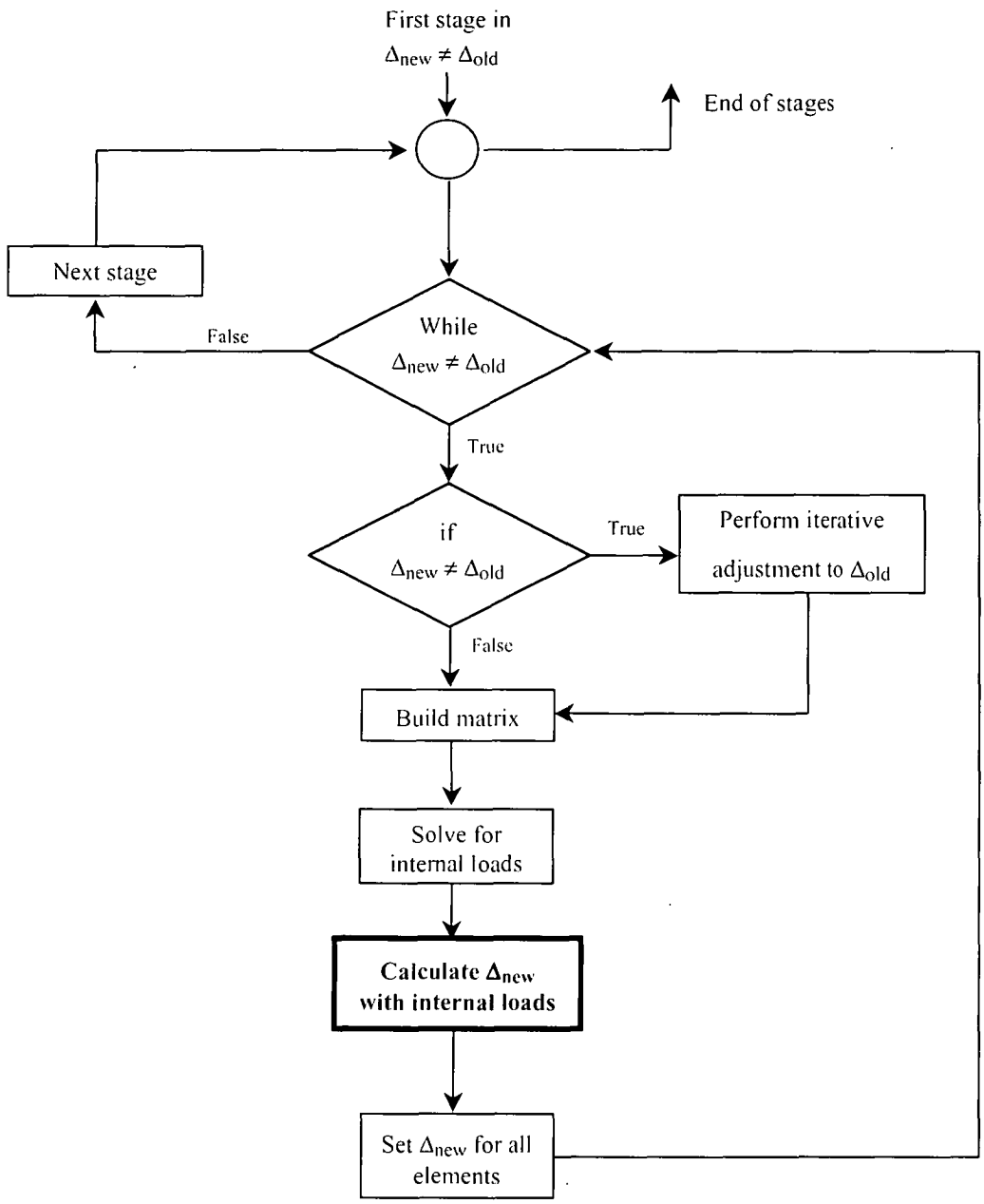


Figure 5.1 Flow Chart for setting up framing analysis

The algorithm represented by the flow chart above sets up the stiffness matrix for the generated internal loads and calculates axial shortening values at each stage in the building cycle. The most important stage in the flow chart operation is the **calculate Δ_{new} with internal loads** stage, shown in bold. The algorithm in the program that performs this step actually calls the non-framing algorithm to calculate all of the axial shortening values used in the stiffness matrix equation (refer bold section of actual algorithm in Appendix A5.2). This is the same algorithm that is used to make the non-framing calculation that is rigorously tested against COLECS. The axial shortening values for a non-framing analysis are shown to be accurate to the highest possible level in the following section of this chapter, thus the remainder of this algorithm represents a new development from which further benchmark testing against independent surveys is not possible.

Testing of the JAVA code in this algorithm for accuracy is also necessary. Fortunately, due to the simplicity of the program language, and the powerful debugging tools this is relatively straightforward and any errors are found during the compiling of the program or during initial trial debugs. Testing the actual workings of the iteration process from within the algorithm and the quantitive results of each line can only be done on a step to step basis whilst a set of calculations is being made. The *System.out.println* lines of code are debugging tools that can be commented in or out to enable outputs of any line of code to be checked during runtime.

The actual calculation procedures of axial shortening with framing action, whilst taking up to four hours computer processing time to complete, has been checked numerous times against COLECS and produces consistent results. The accuracy of the actual quantitive framing adjustment is the only possible non-calibrated area.

5.3 Comparisons

Two sets of data from the Bayoke Tower II in Thailand were used to demonstrate the effect of framing action. Refer Figure 1.1 for tower image. Both data sets differ in two important areas. The first was chosen with relatively small axial differential shortening, while the second set was chosen with the largest axial differential shortening when calculated by COLECS. Refer Appendix A5.2. In addition, Data Set 1 starts with higher axial shortening of the column than the core at lower storeys but finishes with higher axial shortening of the core than the column. Therefore there is a point where the axial shortening values must cross over. This has a significant effect on the outcome of framing action, as is demonstrated later in this chapter.

The data is used to produce axial shortening by two computer programs, COLECS and ASCA. Each analyses the axial shortening of a column and the core element. Results from Data Set 1 have a maximum differential axial shortening of 20.8 % when a non-framing analysis is carried out, whilst Data Set 2 produces significantly larger differential axial shortening of 35.5% between the two elements. Before any framing adjustment is made the results from ASCA are calibrated against COLECS for both sets of data.

All data from an analysis by ASCA is saved as a text file that can be read by MicroSoft Excel® so that charts can be produced[2]. The data from a non-framing analysis of column 1 and the building core is presented below in chart form. Refer Figure 5.2. Superimposed on the same chart are the results for the same data set from COLECS. Actual data for Figure 5.2 can be found in Appendix A5.3. As can be seen from Figure 5.2, both programs produce almost identical results for a non-framing analysis and both curves almost superimpose each other thus some points cannot be seen for both curves.

[2] This is done purely to save time writing complex computer code

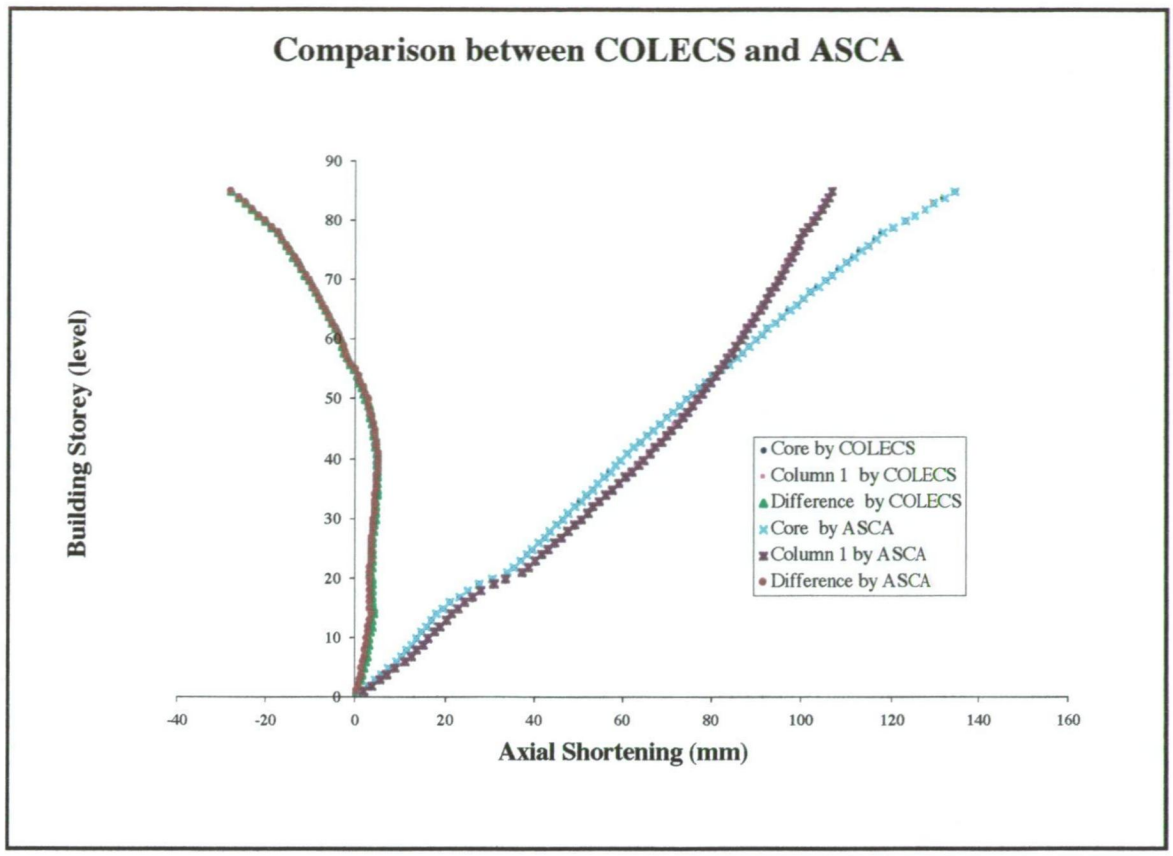


Figure 5.2 Comparisons of non framing results for both COLECS and ASCA for Data Set 1

Figure 5.2 shows that both programs produce data in good agreement with each other. The actual curves are almost superimposed. Some very minor differences do exist in the lower levels, which are demonstrated by Figure 5.3. Here the percentage accuracy between both programs is graphed at each storey for both column 1 and the core. The percentage is calculated by the modular difference between the two results with respect to the output produced by COLECS[3]. The graphs only display percentage values between 80% and 105% to magnify any errors for ease of comparison in chart form.

[3] COLECS is a rigorously tested program with numerous data sets, so this is chosen as the base to calibrate against

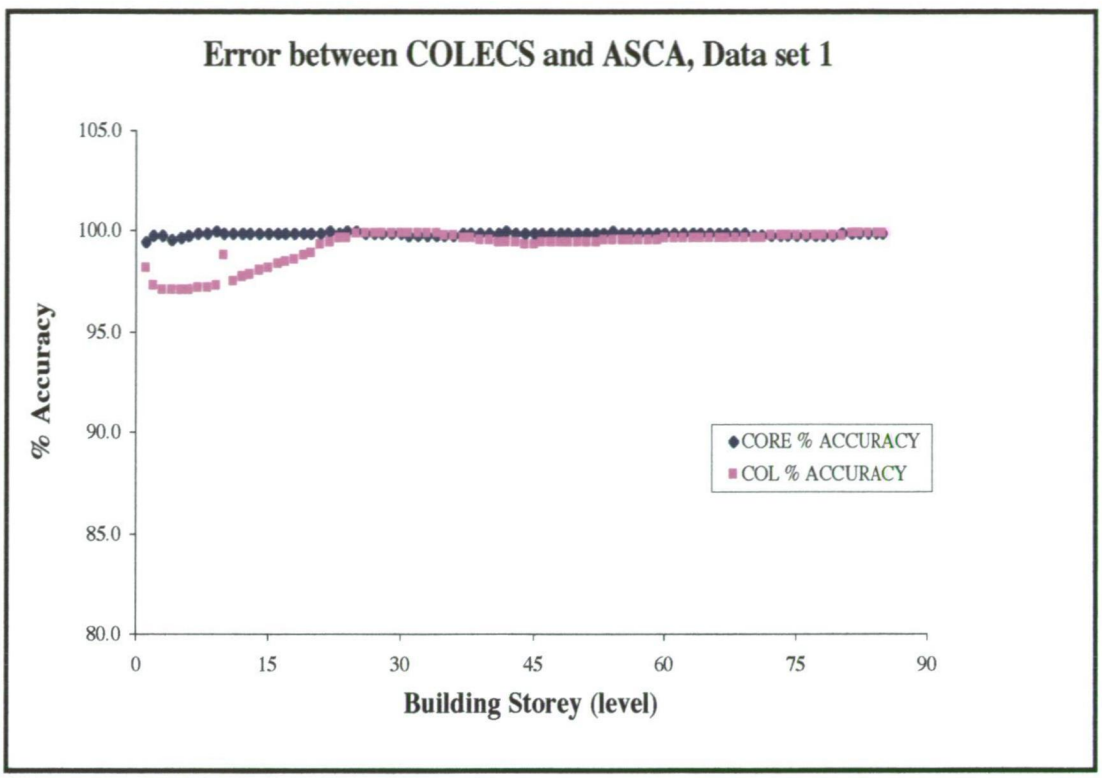


Figure 5.3 Percentage error between ASCA
and COLECS for Data Set 1

Results from COLECS and ASCA for Data Set 2 are also almost identical so the comparison chart (similar to Figure 5.2) is not given. Figure 5.4 represents the percentage error chart for Data Set 2 with percentages calculated in the same way as Figure 5.3 above.

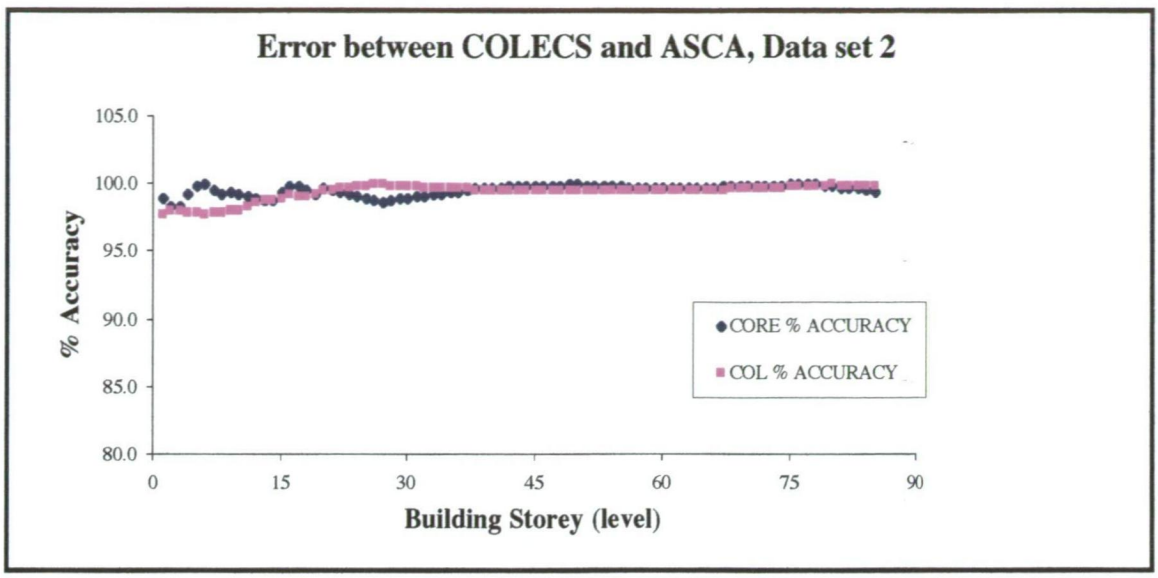


Figure 5.4 Percentage error between ASCA
and COLECS for Data Set 2

What is most noticeable from Figure 5.3 and 5.4 is that the lower 20 to 30 storeys display the greatest variation between the two programs. This can be explained in two ways.

Firstly the calculations of axial shortening are by a cumulative process. An axial shortening calculation for elastic, creep, shrinkage and reinforcing is made for each level at each additional loading stage during the building process. The total axial shortening is the sum of each discrete shortening value for every stage. The lower level may consist of the sum of over 300 small values as opposed to the upper levels that may consist of the sum of less than 10 small values. Thus very small errors are more likely to be magnified in the lower levels.

In addition, the errors above are displayed as percentage errors of the actual value. As the lower levels are all smaller values, the same error in magnitude at the lower levels will show up as a greater percentage based on the smaller values of the lower levels. Nevertheless these errors are still very small and thus it can be concluded that both programs produce very similar results.

5.4 Results from ASCA (2003)

With the program ASCA calibrated against COLECS for axial shortening calculation without considering framing, framing can be introduced to the two data sets in order to critically analyse how framing action affects the axial shortening results.

Each data set used to perform the non-framing comparison above includes one column and the major core of the building. The tributary area floor loads only consider the design strip for the discrete column and the core. The actual building consists of one central core and 20 perimeter columns. Refer Figure 5.5 below;

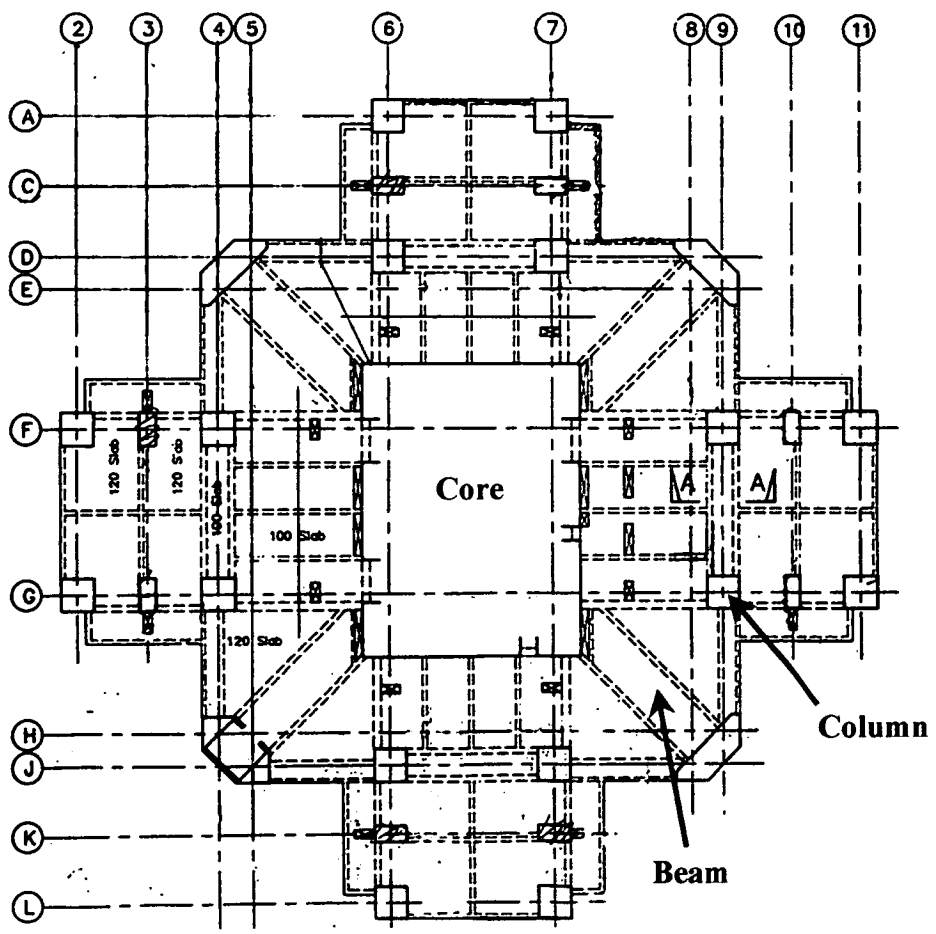


Figure 5.5 Typical lower floor plan, Bayoke Tower II, showing column, core and beams.

For the total axial shortening values calculated by ASCA to be accurate, once framing is introduced, the distribution of shear load transfer to the core needs to reflect the total shear load transfer from all 20 columns onto the core. As we are only considering one column with the core in each data set we need to increase the load transfer onto the core by a factor that will model the load transfer of all 20 columns. In Figure 5.5 above, half the area is carried by 8 columns and the core, while the other half is carried by the remaining 12 columns, therefore each column on average caries 1/19.2 of the shared floor load, or the core shares the total floor area with 19.2 columns. Refer Appendix A5.4 for equation.

The stiffness framing relationship between column and core also needs to be estimated. As discussed in Section 3.2.1, the actual shear load transfer depends upon the column/core connection and the stiffness of the connecting element. Here we will assume a full moment connection and an uncracked concrete beam strip as shown in Figure 5.6. It is important to reiterate here that the mechanics of the load transfer are assumed to be linear and that the beam remains uncracked. This assumption is valid for low moment distribution. Warner (1975) does however consider the effect of a cracked element when he formulates a method to determine framing between two columns with a viscous[4] beam as the connecting element. However this method further complicates an already complicated process and is therefore avoided here in the interest of clarity.

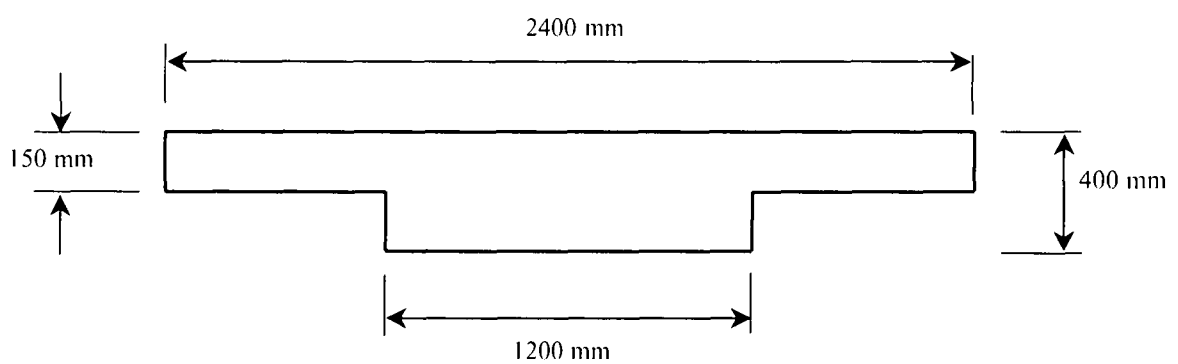


Figure 5.6 Design strip section used to estimate stiffness multiplier

[4] Viscous is a term Warner (1975) uses to describe a cracked connecting beam where the moment of inertia varies with load distribution

5.4.1 Program inputs and stiffness values.

The physical data used to calculate axial shortening values with framing action is given in Appendix A5.5 and is the same as that used for non-framing. The framing stiffness values are calculated based on the geometry in Figures 5.5 and 5.6 above.

I_g for the section in Figure 5.6 used to estimate the stiffness relationship is calculated as

$$I_g = 8.785 \times 10^{-3} \text{ m}^4$$

where I_g is the gross uncracked moment of inertia of the connecting element.

and E_c for 40MPa[5] is estimated using Pauw's equation (1960)

$$E_c = 2400^{1.5} \times 0.043 \times \sqrt{40} = 31975 \text{ MPa}$$

where E_c is estimated modulus of elasticity of the connecting element.

thus

$$E_c I_g = 2.88 \times 10^5 \text{ kN.m}^2$$

Recalling that in the stiffness matrix proposed the framing effect is expressed in terms of the stiffness multiplier reflecting the transfer of shear forces through the framing member

$$F_{j,\|2} = k_{j,\|2} \Delta^D_{j,\|2} \qquad (5.1)$$

For a conventional isotropic beam $k_{1,\|2}$ is $\dfrac{12E_{\|2} I_{\|2}}{\ell_{\|2}^3}$

thus

$$k_{1,\|2} = \frac{12 \times 2.88 \times 10^5}{10^3} = 3.46 \times 10^3 \text{ kN/m} \qquad \text{(column)}$$

The stiffness multiplier at the core is modified to take into account all the connected columns at each floor. Only one column is considered with the core, however the core

[5] This assumes formwork is removed at seven days or greater and no age adjustment is made

experiences load sharing with all the remaining columns in a similar way so the stiffness at the core is increased by 19.2 times as discussed above.

therefore

$$k_{2,12} = 3.46 \times 19.2 = 66.43 \times 10^{3} \text{ kN/m} \qquad \text{(core)}$$

5.4.2 Framing results

Using the stiffness values above, the curves of results for each set of data are shown in Figure 5.7 and 5.8 respectively. The numerical results can be found in Appendix A5.6 In each curve the axial shortening is displayed for both the framed output and the un-framed output and also the differential axial shortening.

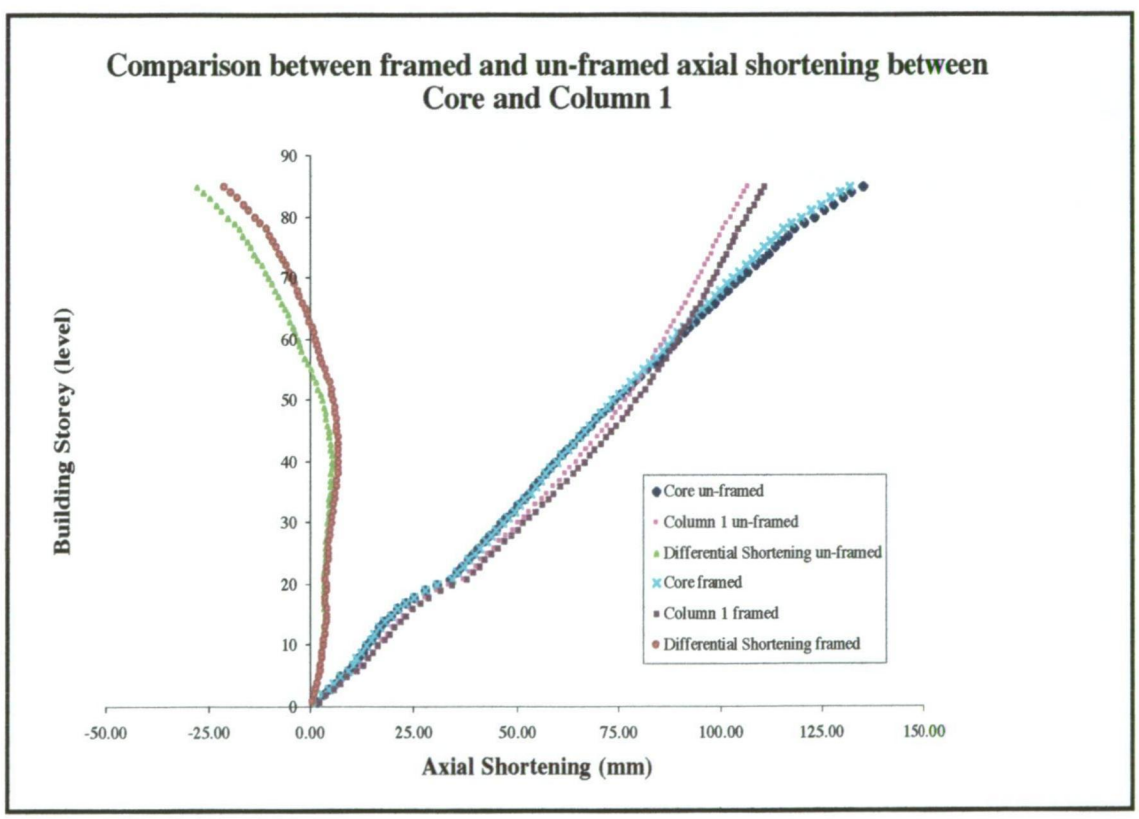


Figure 5.7 Chart of axial shortening with and without framing considerations for Data Set 1

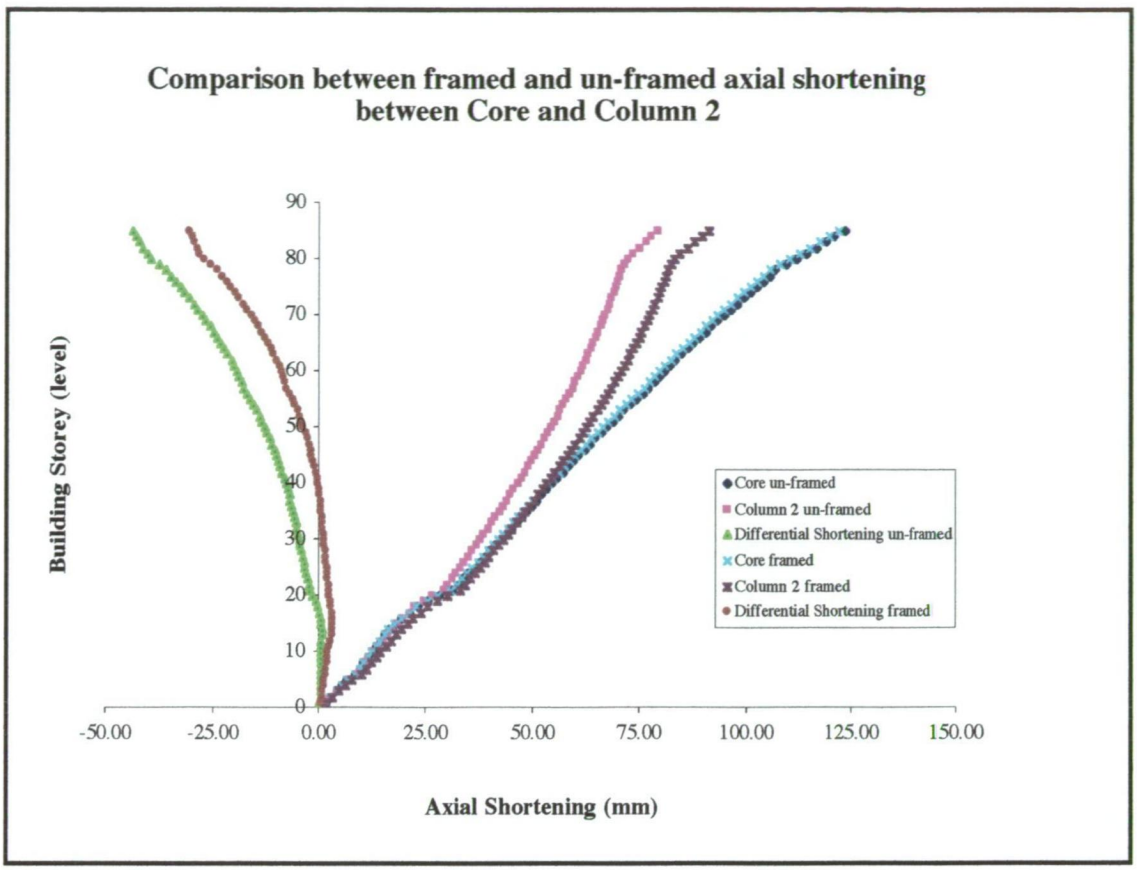


Figure 5.8 Chart of axial shortening with and without framing considerations for Data Set 2

Both charts demonstrate the effect that framing has on the differential axial shortening. In both sets of data the differential axial shortening is reduced when framing is introduced. However the effect on the core, even with the magnified shear transfer, is small. Most of the reduction in differential axial shortening comes from the axial shortening changes in the column due to framing, rather than the core.

This can be put down to the massive difference in size and stiffness between the column and the core. The shear transfer load due to framing action is proportionally much larger in the column than the core, thus the greatest effect of the load on axial shortening is seen in the column. In addition the largest effect of framing is seen in Data Set 2 where the

differential axial shortening is greatest. This is not surprising as the framing action relies on differential displacement when determining shear transfer loads.

Another way to view the data is to consider the effect that framing has on axial shortening values at each level for each data set. This can be expressed in three ways. Consider Table 5.1 displaying the axial shortening data of the 40th storey [6], Data Set 1.

	Axial shortening <u>without</u> framing			**Axial shortening <u>with</u> framing**		
Storey level	Core mm	Column 1 mm	Differential shortening mm	Core mm	Column 1 mm	Differential shortening mm
40	59.42	64.48	-5.05	59.59	66.42	-6.83

Table 5.1 Axial shortening results for the 40th floor with framing action, Data Set 1

Three ways to express the effect that framing has on axial shortening are;

i) The actual difference between framing and non framing axial shortening at each level

ie 59.59 - 59.42 = 0.17 mm (core)

and 66.42 - 64.48 = 1.94 mm (column)

ii) The difference between actual shortening values calculated with framing and without framing expressed as a percentage of actual axial shortening at each level

ie $\frac{59.59-59.42}{59.42}\times 100 = 0.29\%$ (core)

and $\frac{66.42-64.48}{66.42}\times 100 = 2.91\%$ (column)

[6] The 40th floor is chosen arbitrarily to serve as an example only

iii) The difference between differential axial shortening with and without framing, expressed as a percentage of actual axial shortening at each level[7]

$$\text{ie} \quad \frac{-5.05 - -6.83}{59.42} \times 100 = 2.99\% \quad \text{(core)}$$

$$\text{and} \quad \frac{-6.83 - -5.05}{66.42} \times 100 = -2.68\% \quad \text{(column)}$$

Following are charts for both data sets showing how framing affects the axial shortening expressed in the three forms above. Figure 5.9 and 5.10 show the actual difference between framing and non-framing axial shortening at each level for each data set. Figure 5.11 is a combination of Figure 5.9 and 5.10. Figure 5.12 and 5.13 show the alternative ways of expressing the effect of framing action in percentage form.

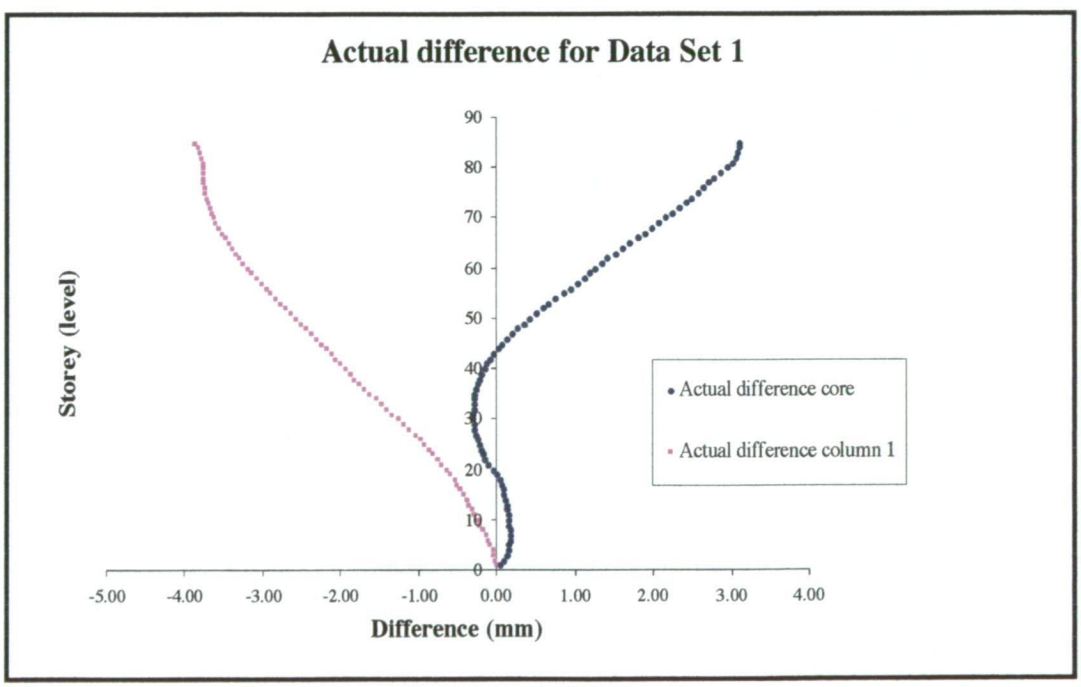


Figure 5.9 Actual shortening differences between framing and non-framing results for Data Set 1

[7] The difference between differential axial shortening is reversed for core and column calculations as the core is the subject of the percentage in the first equation, while the column is the subject in the second

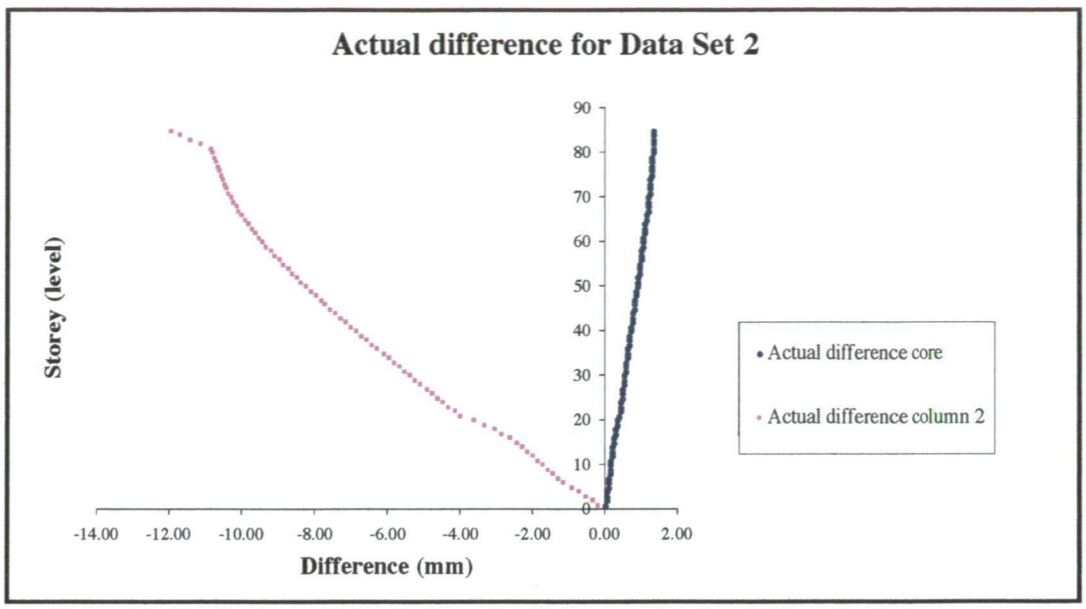


Figure 5.10 Actual shortening differences between framing and non-framing results for Data Set 2

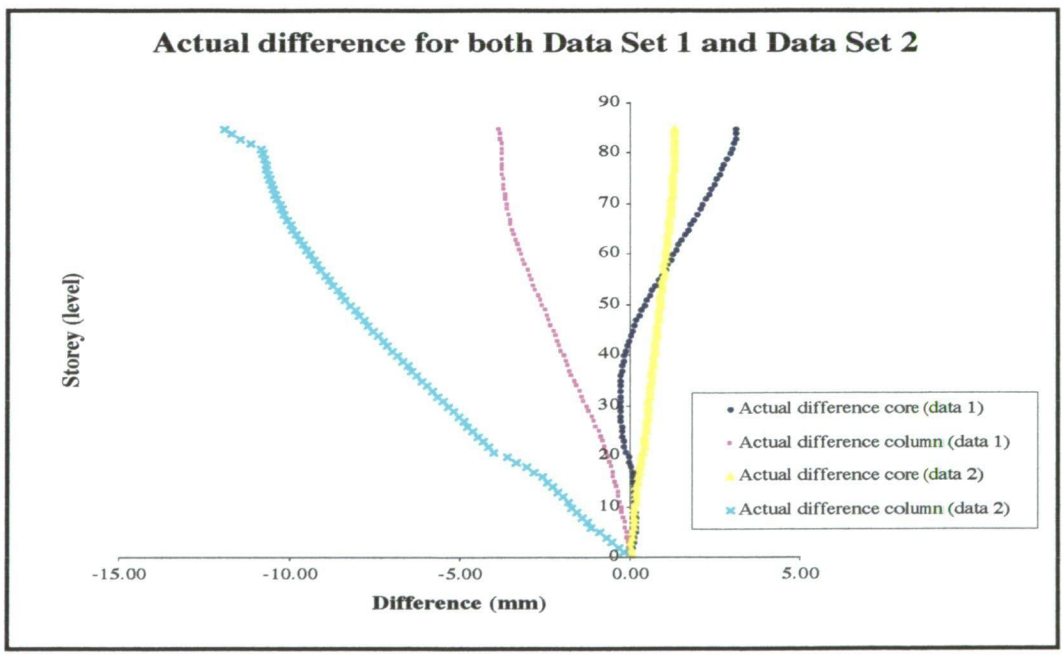


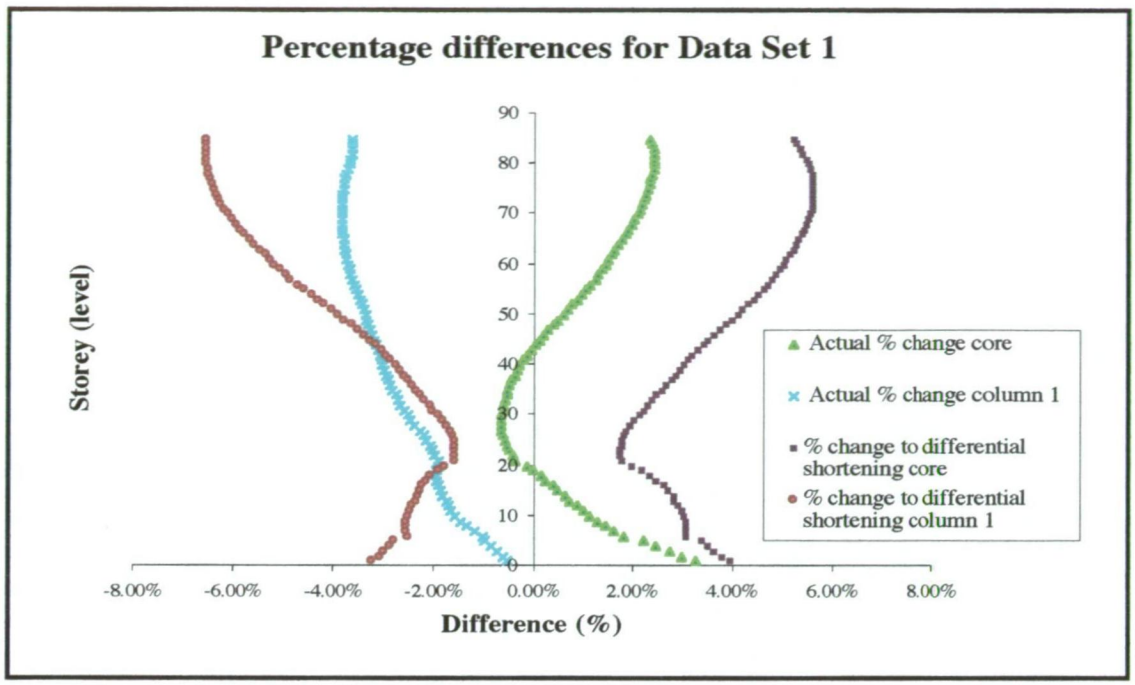
Figure 5.11 Combined actual shortening differences between framing and non-framing results for Data Set 1 and Data Set 2

It is important to note from the charts above that the core experiences the smallest difference to axial shortening when framing is introduced.

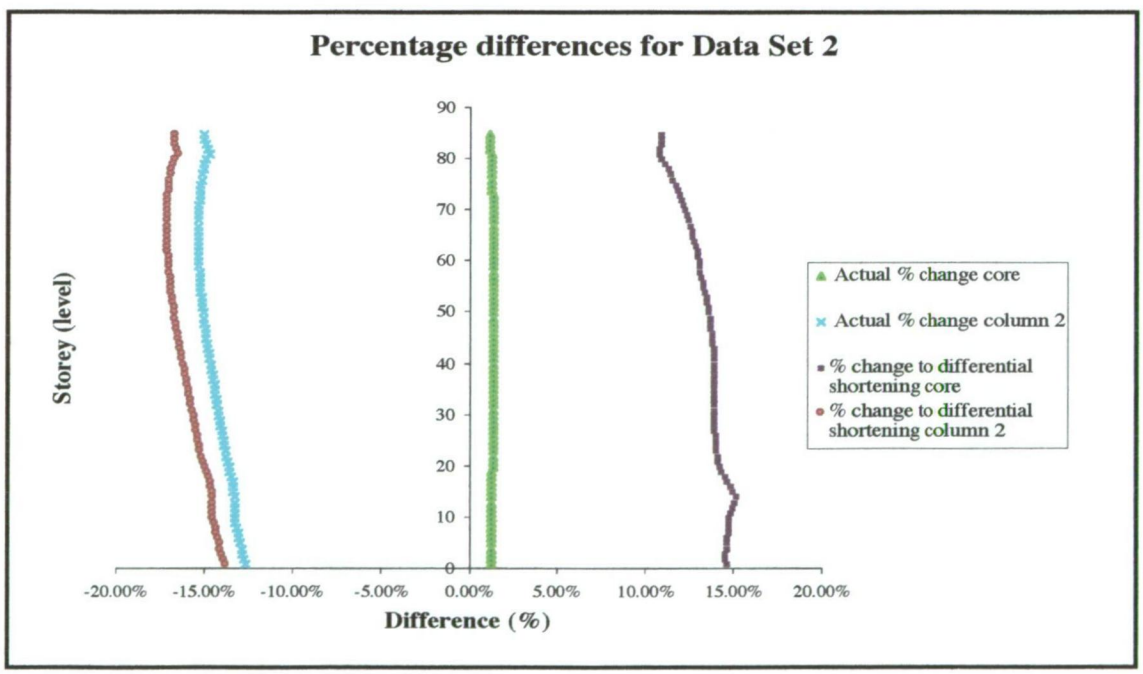
In addition to this, the results from Data Set 2 show that the changes, when framing is introduced, are in the form of a simple curve, whereas in Data Set 1 the curves are more 'S' shaped. In the case of Data Set 2, the core experiences more axial shortening than the column at every storey, thus the curve is quite smooth. In Data Set 1 however, the 'S' shape curve is due to the fact that the column experiences more axial shortening at the lower levels than the core, whereas at the higher levels the core experiences more axial shortening. This means at some point there is no differential axial displacement between the core and column. Although this point may not coincide on the same storey when axial shortening is calculated with framing and without framing, when this cross-over point is approached the actual measured difference between results reduces so an 'S' curve is produced around the cross-over storey.

Another area of importance, which is best displayed in Figure 5.11, is that the actual measured difference between results when framing is introduced in Data Set 2 is greater than for Data Set 1. This is a reflection of the larger axial shortening differential between the column and core for Data Set 2.

By considering changes to axial shortening when framing is introduced in percentage terms, differences are displayed in a normalised manner. Figure 5.12 and 5.13 display the actual difference in percentage terms and differential shortening difference in percentage terms for Data Set 1 and Data Set 2 respectively. The method for determining percentages is explained previously.



5.12 Actual differences and differential differences expressed as a percentage for Data Set 1



5.13 Actual differences and differential differences expressed as a percentage for Data Set 2

Actual percentage differences for each data set clearly show that framing affects the column more than the core. This is most evident in Figure 5.14 for Data Set 2. In Data Set 1 the effect of the cross over between amounts of axial shortening in the column and core is evident in the shape of the core curve where framing decreases axial shortening of the core around the cross over storey.

For both sets of data, the percentage change to differential axial shortening, as a function of the core, is almost equal and opposite when calculated as a function of the column. This is not surprising as the differential difference in both cases is simply equal but opposite. The difference in magnitude is therefore a function of the actual shortening values of the column or core at each storey. Generally they both follow a similar increase at each storey so the magnitude does not differ greatly. However, in Data Set 2 the column axial shortening is distinctly less, so the percentage, as a function of the column, is greater in magnitude than as a function of the core.

Percentage differences in differential axial shortening from Figure 5.12 show the effect of the axial shortening cross over between the column and core in Data Set 1 through the hour glass shape of the each curve around the cross over storey. This is significant, as Figure 5.13 suggests that the effect that framing has on differential axial displacement could almost be predicted as a constant value at every storey. In the case of the core, the difference is + 15% and for the column -17%. Figure 5.12 suggests that framing effects are far more complicated and that predicting a simple constant change to differential axial shortening at each storey is not valid. How each element shortens with respect to the other is as equally important. In addition a cross over between actual shortening values has a major effect on the results once framing is introduced.

5.5 Summary to Chapter

Calibration of the new program ASCA is made with a rigorously tested program, COLECS. The results for the same data sets are shown to be the same within acceptable levels. New results from ASCA are presented for calculations of axial shortening for two data sets with framing action introduced. The results for each data set reveal that framing action causes a change to axial shortening and that this change is a function of the shape of the free axial shortening curves and the degree of differential shortening.

On the basis of the above three important conclusions to draw are;

i) Framing action reduces the differential axial shortening between two elements
ii) The core generally experiences less change to axial shortening than the column
iii) The effect of framing cannot be estimated by a relative percentage adjustment applied uniformly to each storey. It is in fact far more complicated than assumed in previously held theory.

Appendix

A5.1 ASCA code

```
System.out.println("******************** started loop to calc deltas exists************* ");

while ((newDeltasExist == null) || !deltasEqual(oldDeltasExist,newDeltasExist)){

//average old and new deltas
                if (newDeltasExist != null)
                oldDeltasExist=averageDeltas(oldDeltasExist,newDeltasExist);


                 Matrix D = buildD(floorColumnsVector, numColumnsOnFloor,
                oldDeltasExist);\
//System.out.println("D = "); D.print(4,4);

                Matrix KD = KE.times(D);
//System.out.println("KE x D = "); KD.print(4,4);

                Matrix PEKD = PEE.plus(KD);
System.out.println("PEE + KD = "); PEKD.print(4,4);

                Matrix PI = A.solve(PEKD);
System.out.println("PI = "); PI.print(4,4);

//add internal loads to columns
                Vector internalLoads = getInternalLoads(floorColumnsVector,
                                                        numColumnsOnFloor,PI,
                                                        startDay);

//calculate new deltas. A vector of columns will be returned. The delta is an attribute on
the column(tempDeltaExist)

                newDeltasExist = new Vector();

                Vector newDeltasValues =
NonFramingAlgorithm.getInstance().calcColumnDeltaExist(constructedColumns,startDay,endDay,in
ternalLoads);

for (int m = 0; m<newDeltasValues.size(); m++){
                  double d = ((Column)newDeltasValues.elementAt(m)).getTempDeltaExist();
                   newDeltasExist.addElement(new Double(d));
                }

//System.out.println("newDeltasExists    as    calculated    by    calcColumnDeltaExist
"+newDeltasExist);


                if (deltasEqual(oldDeltasExist,newDeltasExist)){
System.out.println("inSetTempCreepPrevious, last loop ");

                  for (int l = 0; l < constructedColumns.size(); l++){
                    Column tempC = (Column)constructedColumns.elementAt(l);
//System.out.println("TempCreepPrevious " + tempC.getTempCreepPrevious());

                    tempC.setTempCreepPrevious(tempC.getTempCreep());
//System.out.println("TempCreepPrevious " + tempC.getTempCreepPrevious());
//System.out.println("TempCreep " + tempC.getTempCreep());

                  }
                }



}// End While
```

A5.2 Column core input data.

Heading meanings are previously defined in A4.3

Data Set 1

STRY NAME	STRY HGHT M	OCPY TIME DAYS	OCPY LOAD KN	CONS TIME DAYS	CONS LOAD KN	CONC GRDE MPA	CONC DENS KG/M3	W/C RAT	CMNT CONT KG/M3	AREA /DPH M2/M	PRMT /WDH M/M	NO. BRS /FR	BASC SHRK COEF	AVG HUM %
LG	4.00	200.	1091.	23.	3194.	50.	2400.	0.45	400.	59.84	150.00	-3.4	500.	70.
			150.	30.	439.	50.	2400.	0.45	400.	1.67	8.00	-30.0	500.	70.
1	4.00	200.	1091.	23.	1935.	50.	2400.	0.45	400.	59.84	150.00	-3.3	500.	70.
			150.	30.	266.	50.	2400.	0.45	400.	1.66	8.00	-29.0	500.	70.
2	4.00	200.	1091.	23.	1935.	50.	2400.	0.45	400.	59.84	150.00	-3.0	500.	70.
			150.	30.	266.	50.	2400.	0.45	400.	1.66	8.00	-29.0	500.	70.
3	4.00	200.	1091.	23.	1935.	50.	2400.	0.45	400.	59.84	150.00	-3.0	500.	70.
			150.	30.	266.	50.	2400.	0.45	400.	1.66	8.00	-29.0	500.	70.
4	4.00	200.	1091.	23.	1935.	50.	2400.	0.45	400.	59.84	150.00	-2.9	500.	70.
			150.	30.	266.	50.	2400.	0.45	400.	1.66	8.00	-29.0	500.	70.
5	5.55	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.6	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.66	8.00	-29.0	500.	70.
6	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.6	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
7	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.5	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
8	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.3	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
9	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.3	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
10	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.3	500.	70.
			60.	30.	316.	50.	2400.	0.45	400.	1.67	8.00	-29.0	500.	70.
11	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.1	500.	70.
			60.	30.	316.	60.	2400.	0.45	400.	1.70	8.00	-27.0	500.	70.
12	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.1	500.	70.
			60.	14.	316.	60.	2400.	0.45	400.	1.70	8.00	-25.0	500.	70.
13	3.10	200.	436.	23.	2301.	50.	2400.	0.45	400.	59.84	150.00	-2.1	500.	70.
			62.	14.	327.	60.	2400.	0.45	400.	1.70	8.00	-25.0	500.	70.
14	3.35	200.	433.	23.	2285.	50.	2400.	0.45	400.	41.55	179.00	-4.7	500.	70.
			62.	14.	327.	60.	2400.	0.45	400.	1.70	8.00	-25.0	500.	70.
15	3.35	200.	1121.	23.	2364.	50.	2400.	0.45	400.	41.55	179.00	-4.5	500.	70.
			155.	14.	327.	60.	2400.	0.45	400.	1.70	8.00	-25.0	500.	70.
15U	4.60	200.	936.	23.	1855.	50.	2400.	0.45	400.	41.55	179.00	-4.4	500.	70.
			155.	14.	307.	60.	2400.	0.45	400.	1.70	8.00	-25.0	500.	70.
16	4.00	200.	936.	23.	1855.	50.	2400.	0.45	400.	41.55	179.00	-4.3	500.	70.
			155.	14.	307.	60.	2400.	0.45	400.	1.70	8.00	-20.0	500.	70.
17	6.10	200.	936.	14.	1855.	50.	2400.	0.45	400.	41.55	179.00	-4.2	500.	70.
			155.	14.	307.	60.	2400.	0.45	400.	1.70	8.00	-20.0	500.	70.
18	6.10	200.	936.	14.	1855.	50.	2400.	0.45	400.	41.55	179.00	-3.9	500.	70.
			155.	14.	307.	60.	2400.	0.45	400.	1.70	8.00	-20.0	500.	70.
19	7.60	200.	936.	14.	4495.	50.	2400.	0.45	400.	41.55	179.00	-3.8	500.	70.
			155.	21.	744.	60.	2400.	0.45	400.	1.69	8.00	-21.0	500.	70.
20	3.10	200.	936.	14.	1798.	50.	2400.	0.45	400.	41.55	179.00	-3.7	500.	70.
			187.	10.	394.	60.	2400.	0.45	400.	1.69	8.00	-17.0	500.	70.
21	3.10	200.	936.	19.	1798.	50.	2400.	0.45	400.	41.55	179.00	-3.7	500.	70.
			187.	10.	394.	60.	2400.	0.45	400.	1.69	8.00	-17.0	500.	70.
22	3.10	200.	936.	14.	1798.	50.	2400.	0.45	400.	41.55	179.00	-3.4	500.	70.
			187.	10.	394.	60.	2400.	0.45	400.	1.69	8.00	-17.0	500.	70.
23	3.10	200.	936.	14.	1798.	50.	2400.	0.45	400.	41.55	179.00	-3.4	500.	70.
			187.	10.	394.	60.	2400.	0.45	400.	1.69	8.00	-17.0	500.	70.
24	3.10	200.	936.	14.	1798.	50.	2400.	0.45	400.	41.55	179.00	-3.4	500.	70.
			187.	10.	394.	60.	2400.	0.45	400.	1.69	8.00	-17.0	500.	70.
25	3.10	200.	936.	14.	1798.	50.	2400.	0.45	400.	41.55	179.00	-3.4	500.	70.
			187.	10.	394.	60.	2400.	0.45	400.	1.69	8.00	-17.0	500.	70.
26	3.10	200.	936.	14.	1798.	60.	2400.	0.45	400.	41.55	179.00	-2.9	500.	70.
			187.	10.	394.	60.	2400.	0.45	400.	1.88	8.00	-11.0	500.	70.
27	3.10	200.	936.	8.	1798.	60.	2400.	0.45	400.	41.55	179.00	-2.9	500.	70.
			187.	10.	394.	60.	2400.	0.45	400.	1.88	8.00	-11.0	500.	70.
28	3.10	200.	936.	8.	1798.	60.	2400.	0.45	400.	41.55	179.00	-2.8	500.	70.
			187.	10.	394.	60.	2400.	0.45	400.	1.88	8.00	-11.0	500.	70.
29	3.10	200.	936.	8.	1798.	60.	2400.	0.45	400.	41.55	179.00	-2.8	500.	70.
			187.	10.	394.	60.	2400.	0.45	400.	1.88	8.00	-11.0	500.	70.
30	3.10	200.	936.	8.	1798.	60.	2400.	0.45	400.	41.55	179.00	-2.4	500.	70.
			187.	10.	394.	60.	2400.	0.45	400.	1.88	8.00	-11.0	500.	70.
31	3.10	200.	936.	8.	1798.	60.	2400.	0.45	400.	41.55	179.00	-2.4	500.	70.
			187.	10.	394.	60.	2400.	0.45	400.	1.88	8.00	-11.0	500.	70.
32	3.10	200.	936.	8.	1798.	60.	2400.	0.45	400.	41.55	179.00	-2.4	500.	70.
			187.	10.	394.	60.	2400.	0.45	400.	1.88	8.00	-9.8	500.	70.
33	3.10	200.	936.	8.	1798.	60.	2400.	0.45	400.	41.55	179.00	-2.0	500.	70.
			187.	10.	394.	60.	2400.	0.45	400.	1.88	8.00	-9.8	500.	70.
34	3.10	200.	936.	8.	1798.	60.	2400.	0.45	400.	41.55	179.00	-2.0	500.	70.
			187.	10.	394.	60.	2400.	0.45	400.	1.88	8.00	-9.8	500.	70.
35	3.10	200.	936.	8.	1798.	60.	2400.	0.45	400.	41.55	179.00	-2.0	500.	70.
			187.	10.	394.	60.	2400.	0.45	400.	1.88	8.00	-9.8	500.	70.
36	3.10	200.	936.	8.	1798.	60.	2400.	0.45	400.	41.55	179.00	-2.0	500.	70.
			195.	10.	477.	60.	2400.	0.45	400.	1.88	8.00	-9.8	500.	70.
37	3.10	200.	936.	8.	1798.	60.	2400.	0.45	400.	41.55	179.00	-1.9	500.	70.
			195.	10.	477.	60.	2400.	0.45	400.	1.88	8.00	-9.8	500.	70.
38	3.10	200.	936.	8.	1798.	60.	2400.	0.45	400.	41.55	179.00	-1.9	500.	70.
			195.	10.	477.	60.	2400.	0.45	400.	1.88	8.00	-9.8	500.	70.
39	3.10	200.	947.	8.	1818.	60.	2400.	0.45	400.	34.18	179.00	-2.8	500.	70.

			195.	10.	477.	60.	2400.	0.45	400.	1.88	8.00	-9.8	500.	70.
40	3.10	200.	947.	8.	1818.	60.	2400.	0.45	400.	34.18	179.00	-2.8	500.	70.
			195.	10.	477.	60.	2400.	0.45	400.	1.88	8.00	-9.8	500.	70.
41	3.10	200.	947.	8.	1818.	60.	2400.	0.45	400.	34.18	179.00	-2.8	500.	70.
			195.	10.	477.	60.	2400.	0.45	400.	1.88	8.00	-8.5	500.	70.
42	3.10	200.	947.	8.	1818.	60.	2400.	0.45	400.	34.18	179.00	-2.7	500.	70.
			195.	10.	477.	60.	2400.	0.45	400.	1.88	8.00	-9.8	500.	70.
43	3.10	200.	947.	8.	1818.	60.	2400.	0.45	400.	34.18	179.00	-2.7	500.	70.
			195.	10.	477.	60.	2400.	0.45	400.	1.88	8.00	-8.5	500.	70.
44	3.10	200.	947.	8.	1818.	60.	2400.	0.45	400.	34.18	179.00	-2.6	500.	70.
			195.	10.	477.	60.	2400.	0.45	400.	1.88	8.00	-8.5	500.	70.
45	3.10	200.	947.	8.	1818.	60.	2400.	0.45	400.	34.18	179.00	-2.6	500.	70.
			195.	10.	477.	60.	2400.	0.45	400.	1.88	8.00	-8.5	500.	70.
46	3.10	200.	947.	8.	1818.	60.	2400.	0.45	400.	34.18	179.00	-1.7	500.	70.
			195.	10.	477.	60.	2400.	0.45	400.	1.88	8.00	-8.5	500.	70.
47	3.10	200.	947.	8.	1818.	60.	2400.	0.45	400.	34.18	179.00	-1.6	500.	70.
			195.	10.	477.	60.	2400.	0.45	400.	1.88	8.00	-8.5	500.	70.
48	3.10	200.	947.	8.	1818.	60.	2400.	0.45	400.	34.18	179.00	-1.1	500.	70.
			195.	10.	477.	60.	2400.	0.45	400.	1.88	8.00	-8.5	500.	70.
49	3.10	200.	947.	8.	1818.	60.	2400.	0.45	400.	34.18	179.00	-1.0	500.	70.
			195.	10.	477.	60.	2400.	0.45	400.	1.88	8.00	-8.5	500.	70.
50	3.10	200.	947.	8.	1818.	60.	2400.	0.45	400.	34.18	179.00	-1.0	500.	70.
			195.	10.	477.	60.	2400.	0.45	400.	1.88	8.00	-8.5	500.	70.
51	3.10	200.	947.	8.	1818.	60.	2400.	0.45	400.	34.18	179.00	-1.0	500.	70.
			195.	10.	477.	60.	2400.	0.45	400.	1.88	8.00	-7.9	500.	70.
52	3.10	200.	1515.	8.	2091.	60.	2400.	0.45	400.	34.18	179.00	-0.2	500.	70.
			232.	10.	349.	60.	2400.	0.45	400.	1.88	8.00	-7.9	500.	70.
53	3.10	200.	1029.	8.	1975.	60.	2400.	0.45	400.	31.10	179.00	-0.3	500.	70.
			134.	10.	286.	60.	2400.	0.45	400.	1.88	8.00	-7.9	500.	70.
54	3.10	200.	1047.	8.	2010.	60.	2400.	0.45	400.	31.10	179.00	-0.3	500.	70.
			124.	10.	265.	60.	2400.	0.45	400.	1.88	8.00	-7.9	500.	70.
55	3.10	200.	1047.	8.	2010.	60.	2400.	0.45	400.	31.10	179.00	-0.1	500.	70.
			113.	10.	245.	60.	2400.	0.45	400.	1.88	8.00	-7.9	500.	70.
56	3.10	200.	1047.	8.	2010.	60.	2400.	0.45	400.	31.10	179.00	-2.3	500.	70.
			113.	10.	245.	60.	2400.	0.45	400.	1.88	8.00	-7.9	500.	70.
57	3.10	200.	1047.	8.	2010.	60.	2400.	0.45	400.	31.10	179.00	-2.2	500.	70.
			113.	10.	245.	60.	2400.	0.45	400.	1.88	8.00	-7.9	500.	70.
58	3.10	200.	1047.	8.	2010.	60.	2400.	0.45	400.	31.10	179.00	-2.0	500.	70.
			113.	10.	245.	60.	2400.	0.45	400.	1.88	8.00	-7.9	500.	70.
59	3.10	200.	1047.	8.	2010.	60.	2400.	0.45	400.	31.10	179.00	-1.9	500.	70.
			113.	10.	245.	60.	2400.	0.45	400.	1.88	8.00	-7.9	500.	70.
60	3.10	200.	1047.	8.	2010.	60.	2400.	0.45	400.	31.10	179.00	-1.9	500.	70.
			113.	10.	245.	60.	2400.	0.45	400.	1.88	8.00	-7.9	500.	70.
61	3.10	200.	1060.	8.	2035.	60.	2400.	0.45	400.	23.00	179.00	-3.0	500.	70.
			113.	10.	245.	50.	2400.	0.45	400.	1.92	8.00	-5.6	500.	70.
62	3.10	200.	1060.	8.	2035.	60.	2400.	0.45	400.	23.00	179.00	-3.0	500.	70.
			113.	10.	245.	50.	2400.	0.45	400.	1.92	8.00	-5.6	500.	70.
63	3.10	200.	1060.	8.	2035.	60.	2400.	0.45	400.	23.00	179.00	-3.0	500.	70.
			113.	10.	245.	50.	2400.	0.45	400.	1.92	8.00	-5.6	500.	70.
64	3.10	200.	1060.	8.	2035.	60.	2400.	0.45	400.	23.00	179.00	-2.8	500.	70.
			113.	10.	245.	50.	2400.	0.45	400.	1.92	8.00	-5.6	500.	70.
65	3.10	200.	1060.	8.	2035.	60.	2400.	0.45	400.	23.00	179.00	-2.6	500.	70.
			113.	10.	245.	50.	2400.	0.45	400.	1.92	8.00	-5.6	500.	70.
66	3.10	200.	1060.	8.	2035.	60.	2400.	0.45	400.	23.00	179.00	-2.6	500.	70.
			113.	10.	245.	50.	2400.	0.45	400.	1.92	8.00	-5.6	500.	70.
67	3.10	200.	1060.	8.	2035.	60.	2400.	0.45	400.	23.00	179.00	-2.4	500.	70.
			113.	10.	245.	50.	2400.	0.45	400.	1.92	8.00	-5.6	500.	70.
68	3.10	200.	1060.	8.	2035.	60.	2400.	0.45	400.	23.00	179.00	-2.4	500.	70.
			113.	10.	245.	50.	2400.	0.45	400.	1.92	8.00	-5.6	500.	70.
69	3.10	200.	1060.	8.	2035.	50.	2400.	0.45	400.	23.00	179.00	-2.1	500.	70.
			113.	10.	245.	50.	2400.	0.45	400.	1.92	8.00	-5.6	500.	70.
70	3.10	200.	1060.	8.	2035.	50.	2400.	0.45	400.	23.00	179.00	-1.9	500.	70.
			113.	10.	245.	50.	2400.	0.45	400.	1.92	8.00	-5.6	500.	70.
71	3.10	200.	1060.	8.	2035.	50.	2400.	0.45	400.	23.00	179.00	-1.9	500.	70.
			113.	10.	245.	50.	2400.	0.45	400.	1.95	8.00	-4.2	500.	70.
72	3.10	200.	1060.	8.	2035.	50.	2400.	0.45	400.	23.00	179.00	-1.9	500.	70.
			113.	10.	245.	50.	2400.	0.45	400.	1.95	8.00	-4.2	500.	70.
73	3.10	200.	1060.	8.	2035.	50.	2400.	0.45	400.	23.00	179.00	-1.6	500.	70.
			113.	10.	245.	50.	2400.	0.45	400.	1.95	8.00	-4.2	500.	70.
74	3.10	200.	1060.	8.	2035.	50.	2400.	0.45	400.	23.00	179.00	-1.6	500.	70.
			113.	10.	245.	50.	2400.	0.45	400.	1.95	8.00	-4.2	500.	70.
75	3.10	200.	1060.	8.	2035.	50.	2400.	0.45	400.	23.00	179.00	-1.5	500.	70.
			113.	10.	245.	50.	2400.	0.45	400.	1.95	8.00	-4.2	500.	70.
76	3.10	200.	425.	8.	2141.	50.	2400.	0.45	400.	22.13	172.00	-1.7	500.	70.
			45.	10.	255.	50.	2400.	0.45	400.	1.95	8.00	-4.2	500.	70.
77	5.00	200.	454.	8.	2290.	50.	2400.	0.45	400.	22.13	172.00	-1.7	500.	70.
			45.	10.	255.	50.	2400.	0.45	400.	1.95	8.00	-4.2	500.	70.
78	5.00	200.	2279.	8.	8418.	50.	2400.	0.45	400.	22.13	172.00	-1.7	500.	70.
			45.	10.	363.	50.	2400.	0.45	400.	1.95	8.00	-4.2	500.	70.
79	5.00	200.	953.	8.	1601.	50.	2400.	0.45	400.	22.13	172.00	-1.3	500.	70.
			83.	10.	340.	50.	2400.	0.45	400.	1.95	8.00	-4.2	500.	70.
80	5.00	200.	318.	8.	1601.	50.	2400.	0.45	400.	22.13	172.00	-1.3	500.	70.
			36.	10.	333.	50.	2400.	0.45	400.	1.95	8.00	-4.2	500.	70.
81	5.00	200.	318.	8.	1601.	50.	2400.	0.45	400.	22.13	172.00	-1.3	500.	70.
			24.	10.	143.	50.	2400.	0.45	400.	1.95	8.00	-4.2	500.	70.
82	5.00	200.	318.	8.	1601.	50.	2400.	0.45	400.	22.13	172.00	-1.1	500.	70.
			24.	10.	143.	50.	2400.	0.45	400.	1.95	8.00	-4.2	500.	70.
83	5.00	200.	41993.	8.	33745.	50.	2400.	0.45	400.	16.41	128.00	-1.5	500.	70.
			73.	10.	148.	50.	2400.	0.45	400.	1.95	8.00	-4.2	500.	70.

Data Set 2

STRY NAME	STRY HGHT M	OCPY TIME DAYS	OCPY LOAD KN	CONS TIME DAYS	CONS LOAD KN	CONC GRDE MPA	CONC DENS KG/M3	W/C RAT	CMNT CONT KG/M3	AREA /DPH M2/M	PRMT /WDH M/M	NO BRS /%R	BASC SHRK COEF	AVG HUM %
LG	4.00	200.	1091.	23.	3194.	70.	2400.	0.45	400.	59.84	150.00	-3.4	350.	50.
			103.	30.	607.	70.	2400.	0.45	400.	5.21	15.90	-7.5	350.	50.
1	4.00	200.	1091.	23.	1935.	70.	2400.	0.45	400.	59.84	150.00	-3.3	350.	50.
			103.	30.	367.	70.	2400.	0.45	400.	5.21	15.90	-7.5	350.	50.
2	4.00	200.	1091.	23.	1935.	70.	2400.	0.45	400.	59.84	150.00	-3.0	350.	50.
			103.	30.	367.	70.	2400.	0.45	400.	5.21	15.90	-7.5	350.	50.
3	4.00	200.	1091.	23.	1935.	70.	2400.	0.45	400.	59.84	150.00	-3.0	350.	50.
			103.	30.	367.	70.	2400.	0.45	400.	5.21	15.90	-7.5	350.	50.
4	4.00	200.	1091.	23.	1935.	70.	2400.	0.45	400.	59.84	150.00	-2.9	350.	50.
			103.	30.	367.	70.	2400.	0.45	400.	5.21	15.90	-7.5	350.	50.
5	5.55	200.	436.	23.	2301.	70.	2400.	0.45	400.	59.84	150.00	-2.6	350.	50.
			41.	30.	436.	70.	2400.	0.45	400.	5.21	15.90	-7.5	350.	50.
6	3.10	200.	436.	23.	2301.	70.	2400.	0.45	400.	59.84	150.00	-2.6	350.	50.
			41.	30.	436.	70.	2400.	0.45	400.	5.21	15.90	-7.5	350.	50.
7	3.10	200.	436.	23.	2301.	70.	2400.	0.45	400.	59.84	150.00	-2.5	350.	50.
			41.	30.	436.	70.	2400.	0.45	400.	5.21	15.90	-7.5	350.	50.
8	3.10	200.	436.	23.	2301.	70.	2400.	0.45	400.	59.84	150.00	-2.3	350.	50.
			41.	30.	436.	70.	2400.	0.45	400.	5.21	15.90	-7.5	350.	50.
9	3.10	200.	436.	23.	2301.	70.	2400.	0.45	400.	59.84	150.00	-2.3	350.	50.
			41.	30.	436.	70.	2400.	0.45	400.	5.21	15.90	-7.5	350.	50.
10	3.10	200.	436.	23.	2301.	70.	2400.	0.45	400.	59.84	150.00	-2.3	350.	50.
			41.	30.	436.	70.	2400.	0.45	400.	5.42	15.90	-3.3	350.	50.
11	3.10	200.	436.	23.	2301.	70.	2400.	0.45	400.	59.84	150.00	-2.1	350.	50.
			41.	30.	436.	70.	2400.	0.45	400.	5.42	15.90	-3.3	350.	50.
12	3.10	200.	436.	23.	2301.	70.	2400.	0.45	400.	59.84	150.00	-2.1	350.	50.
			41.	14.	436.	70.	2400.	0.45	400.	5.42	15.90	-3.3	350.	50.
13	3.10	200.	436.	23.	2301.	70.	2400.	0.45	400.	59.84	150.00	-2.1	350.	50.
			41.	14.	436.	70.	2400.	0.45	400.	5.45	15.90	-2.8	350.	50.
14	3.35	200.	433.	23.	2285.	80.	2400.	0.45	400.	41.55	179.00	-4.7	350.	50.
			41.	14.	436.	70.	2400.	0.45	400.	5.45	15.90	-2.8	350.	50.
15	3.35	200.	1121.	23.	2364.	80.	2400.	0.45	400.	41.55	179.00	-4.5	350.	50.
			103.	14.	436.	70.	2400.	0.45	400.	5.45	15.90	-2.8	350.	50.
15U	4.60	200.	936.	23.	1855.	80.	2400.	0.45	400.	41.55	179.00	-4.4	350.	50.
			103.	14.	410.	70.	2400.	0.45	400.	5.45	15.90	-2.8	350.	50.
16	4.00	200.	936.	23.	1855.	80.	2400.	0.45	400.	41.55	179.00	-4.3	350.	50.
			103.	14.	410.	70.	2400.	0.45	400.	5.46	15.90	-2.5	350.	50.
17	6.10	200.	936.	14.	1855.	80.	2400.	0.45	400.	41.55	179.00	-4.2	350.	50.
			103.	14.	410.	70.	2400.	0.45	400.	5.46	15.90	-2.5	350.	50.
18	6.10	200.	936.	14.	1855.	80.	2400.	0.45	400.	41.55	179.00	-3.9	350.	50.
			103.	14.	410.	70.	2400.	0.45	400.	5.46	15.90	-2.5	350.	50.
19	7.60	200.	936.	14.	4495.	80.	2400.	0.45	400.	41.55	179.00	-3.8	350.	50.
			103.	21.	992.	70.	2400.	0.45	400.	5.49	15.90	-2.1	350.	50.
20	3.10	200.	936.	14.	1798.	80.	2400.	0.45	400.	41.55	179.00	-3.7	350.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.49	15.90	-2.1	350.	50.
21	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-3.7	350.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.49	15.90	-2.1	350.	50.
22	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-3.4	350.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.49	15.90	-2.0	350.	50.
23	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-3.4	350.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.49	15.90	-2.0	350.	50.
24	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-3.4	350.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.49	15.90	-2.0	350.	50.
25	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-3.4	350.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.49	15.90	-2.0	350.	50.
26	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-2.9	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.49	15.90	-2.0	350.	50.
27	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-2.9	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.49	15.90	-2.0	350.	50.
28	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-2.8	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.49	15.90	-2.0	350.	50.
29	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-2.8	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.49	15.90	-2.0	350.	50.
30	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-2.4	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.49	15.90	-2.0	350.	50.
31	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-2.4	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.50	15.90	-1.9	350.	50.
32	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-2.4	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.50	15.90	-1.9	350.	50.
33	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-2.0	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.50	15.90	-1.9	350.	50.
34	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-2.0	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.50	15.90	-1.9	350.	50.
35	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-2.0	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.50	15.90	-1.9	350.	50.
36	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-2.0	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.50	15.90	-1.9	350.	50.
37	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-1.9	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.50	15.90	-1.9	350.	50.
38	3.10	200.	936.	8.	1798.	80.	2400.	0.45	400.	41.55	179.00	-1.9	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.50	15.90	-1.9	350.	50.
39	3.10	200.	947.	8.	1818.	80.	2400.	0.45	400.	34.18	179.00	-2.8	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.5	350.	50.
40	3.10	200.	947.	8.	1818.	80.	2400.	0.45	400.	34.18	179.00	-2.8	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.5	350.	50.
41	3.10	200.	947.	8.	1818.	80.	2400.	0.45	400.	34.18	179.00	-2.8	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.5	350.	50.
42	3.10	200.	947.	8.	1818.	80.	2400.	0.45	400.	34.18	179.00	-2.7	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.5	350.	50.
43	3.10	200.	947.	8.	1818.	80.	2400.	0.45	400.	34.18	179.00	-2.7	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.5	350.	50.

44	3.10	200.	947.	8.	1818.	80.	2400.	0.45	400.	34.18	179.00	-2.6	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.5	350.	50.
45	3.10	200.	947.	8.	1818.	80.	2400.	0.45	400.	34.18	179.00	-2.6	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.5	350.	50.
46	3.10	200.	947.	8.	1818.	80.	2400.	0.45	400.	34.18	179.00	-1.7	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.5	350.	50.
47	3.10	200.	947.	8.	1818.	80.	2400.	0.45	400.	34.18	179.00	-1.6	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.5	350.	50.
48	3.10	200.	947.	8.	1818.	80.	2400.	0.45	400.	34.18	179.00	-1.1	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.5	350.	50.
49	3.10	200.	947.	8.	1818.	80.	2400.	0.45	400.	34.18	179.00	-1.0	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.5	350.	50.
50	3.10	200.	947.	8.	1818.	80.	2400.	0.45	400.	34.18	179.00	-1.0	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.5	350.	50.
51	3.10	200.	947.	8.	1818.	80.	2400.	0.45	400.	34.18	179.00	-1.0	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
52	3.10	200.	1515.	8.	2091.	80.	2400.	0.45	400.	34.18	179.00	-0.2	300.	50.
			80.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
53	3.10	200.	1029.	8.	1975.	80.	2400.	0.45	400.	31.10	179.00	-0.3	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
54	3.10	200.	1047.	8.	2010.	80.	2400.	0.45	400.	31.10	179.00	-0.3	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
55	3.10	200.	1047.	8.	2010.	80.	2400.	0.45	400.	31.10	179.00	-0.1	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
56	3.10	200.	1047.	8.	2010.	80.	2400.	0.45	400.	31.10	179.00	-2.3	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
57	3.10	200.	1047.	8.	2010.	80.	2400.	0.45	400.	31.10	179.00	-2.2	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
58	3.10	200.	1047.	8.	2010.	80.	2400.	0.45	400.	31.10	179.00	-2.0	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
59	3.10	200.	1047.	8.	2010.	80.	2400.	0.45	400.	31.10	179.00	-1.9	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
60	3.10	200.	1047.	8.	2010.	80.	2400.	0.45	400.	31.10	179.00	-1.9	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
61	3.10	200.	1060.	8.	2035.	80.	2400.	0.45	400.	23.00	179.00	-3.0	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
62	3.10	200.	1060.	8.	2035.	80.	2400.	0.45	400.	23.00	179.00	-3.0	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
63	3.10	200.	1060.	8.	2035.	80.	2400.	0.45	400.	23.00	179.00	-3.0	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
64	3.10	200.	1060.	8.	2035.	80.	2400.	0.45	400.	23.00	179.00	-2.8	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
65	3.10	200.	1060.	8.	2035.	80.	2400.	0.45	400.	23.00	179.00	-2.6	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
66	3.10	200.	1060.	8.	2035.	80.	2400.	0.45	400.	23.00	179.00	-2.6	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
67	3.10	200.	1060.	8.	2035.	80.	2400.	0.45	400.	23.00	179.00	-2.4	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
68	3.10	200.	1060.	8.	2035.	80.	2400.	0.45	400.	23.00	179.00	-2.4	300.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
69	3.10	200.	1060.	8.	2035.	70.	2400.	0.45	400.	23.00	179.00	-2.1	350.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
70	3.10	200.	1060.	8.	2035.	70.	2400.	0.45	400.	23.00	179.00	-1.9	350.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
71	3.10	200.	1060.	8.	2035.	70.	2400.	0.45	400.	23.00	179.00	-1.9	350.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
72	3.10	200.	1060.	8.	2035.	70.	2400.	0.45	400.	23.00	179.00	-1.9	350.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
73	3.10	200.	1060.	8.	2035.	70.	2400.	0.45	400.	23.00	179.00	-1.6	350.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
74	3.10	200.	1060.	8.	2035.	70.	2400.	0.45	400.	23.00	179.00	-1.6	350.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
75	3.10	200.	1060.	8.	2035.	70.	2400.	0.45	400.	23.00	179.00	-1.5	350.	50.
			50.	10.	231.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
76	3.10	200.	425.	8.	2141.	70.	2400.	0.45	400.	22.13	172.00	-1.7	350.	50.
			21.	10.	236.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
77	5.00	200.	454.	8.	2290.	70.	2400.	0.45	400.	22.13	172.00	-1.7	350.	50.
			23.	10.	236.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
78	5.00	200.	2279.	8.	8418.	70.	2400.	0.45	400.	22.13	172.00	-1.7	350.	50.
			26.	10.	256.	70.	2400.	0.45	400.	5.52	15.90	-1.3	350.	50.
79	5.00	200.	953.	8.	1601.	70.	2400.	0.45	400.	22.13	172.00	-1.3	350.	50.
			96.	10.	236.	70.	2400.	0.45	400.	2.76	15.90	-2.5	350.	50.
80	5.00	200.	318.	8.	1601.	70.	2400.	0.45	400.	22.13	172.00	-1.3	350.	50.
			0.	10.	0.	50.	0.	0.45	400.	1.95	8.00	-4.2	0.	50.
81	5.00	200.	318.	8.	1601.	70.	2400.	0.45	400.	22.13	172.00	-1.3	350.	50.
			0.	10.	0.	50.	0.	0.45	400.	1.95	8.00	-4.2	0.	50.
82	5.00	200.	318.	8.	1601.	70.	2400.	0.45	400.	22.13	172.00	-1.1	350.	50.
			0.	10.	0.	50.	0.	0.45	400.	1.95	8.00	-4.2	0.	50.
83	5.00	200.	41993.	8.	33745.	70.	2400.	0.45	400.	16.41	128.00	-1.5	350.	50.
			0.	10.	0.	50.	0.	0.45	400.	1.95	8.00	-4.2	0.	50.

A5.3 Results of comparison between COLECS and ASCA

Data Set 1

Axial Shortening calculated by COLECS				Axial Shortening calculated by ASCA		
Storey	Core Shortening	Column Shortening	Differential Shortening	Core Shortening	Column Shortening	Differential Shortening
	mm	mm	mm	mm	mm	mm
1	1.45	1.79	-0.34	1.46	1.76	-0.30
2	2.9	3.61	-0.71	2.90	3.51	-0.62
3	4.35	5.41	-1.06	4.34	5.25	-0.91
4	5.79	7.18	-1.39	5.77	6.98	-1.21
5	7.22	8.94	-1.72	7.20	8.68	-1.48
6	9.22	11.36	-2.14	9.21	11.03	-1.82
7	10.32	12.68	-2.36	10.32	12.33	-2.01
8	11.42	14	-2.58	11.42	13.62	-2.20
9	12.52	15.3	-2.78	12.52	14.90	-2.38
10	13.61	16.358	-2.748	13.61	16.17	-2.55
11	14.69	17.86	-3.17	14.70	17.43	-2.73
12	15.77	19.14	-3.37	15.78	18.71	-2.92
13	16.85	20.47	-3.62	16.86	20.05	-3.19
14	17.91	21.79	-3.88	17.93	21.38	-3.45
15	19.44	23.21	-3.77	19.45	22.80	-3.35
16	20.98	24.6	-3.62	20.99	24.21	-3.21
17	23.08	26.5	-3.42	23.09	26.12	-3.03
18	24.9	28.36	-3.46	24.92	27.99	-3.07
19	27.67	31.16	-3.49	27.68	30.81	-3.13
20	30.46	33.92	-3.46	30.48	33.59	-3.11
21	33.9	37.23	-3.33	33.91	37.01	-3.10
22	35.29	38.72	-3.43	35.29	38.54	-3.25
23	36.67	40.18	-3.51	36.66	40.05	-3.39
24	38.06	41.63	-3.57	38.06	41.53	-3.48
25	39.44	43.05	-3.61	39.44	43.00	-3.56
26	40.81	44.45	-3.64	40.82	44.45	-3.63
27	42.17	45.83	-3.66	42.18	45.87	-3.69
28	43.51	47.35	-3.84	43.55	47.37	-3.83
29	44.85	48.84	-3.99	44.90	48.86	-3.96
30	46.19	50.31	-4.12	46.24	50.32	-4.08
31	47.51	51.76	-4.25	47.57	51.75	-4.19
32	48.85	53.18	-4.33	48.93	53.17	-4.24
33	50.19	54.58	-4.39	50.27	54.56	-4.28
34	51.51	56.03	-4.52	51.61	56.05	-4.44
35	52.86	57.45	-4.59	52.96	57.52	-4.56
36	54.2	58.85	-4.65	54.27	58.96	-4.69
37	55.52	60.22	-4.7	55.57	60.38	-4.81
38	56.83	61.58	-4.75	56.88	61.77	-4.89
39	58.14	62.9	-4.76	58.16	63.14	-4.98
40	59.44	64.2	-4.76	59.42	64.48	-5.05
41	60.9	65.47	-4.57	60.89	65.79	-4.90
42	62.35	66.71	-4.36	62.35	67.07	-4.72
43	63.78	68	-4.22	63.79	68.36	-4.57

Axial Shortening calculated by COLECS				Axial Shortening calculated by ASCA		
Storey	Core Shortening	Column 1 Shortening	Differential Shortening	Core Shortening	Column 1 Shortening	Differential Shortening
	mm	mm	mm	mm	mm	mm
44	65.21	69.18	-3.97	65.22	69.58	-4.36
45	66.63	70.41	-3.78	66.65	70.81	-4.15
46	68.05	71.62	-3.57	68.08	72.01	-3.93
47	69.45	72.79	-3.34	69.49	73.17	-3.69
48	70.93	73.93	-3	71.00	74.31	-3.31
49	72.42	75.05	-2.63	72.50	75.42	-2.93
50	73.95	76.14	-2.19	73.88	76.50	-2.62
51	75.48	77.19	-1.71	75.43	77.55	-2.13
52	77	78.22	-1.22	76.96	78.57	-1.61
53	78.5	79.25	-0.75	78.48	79.60	-1.12
54	80.1	80.25	-0.15	80.10	80.59	-0.49
55	81.81	81.22	0.59	81.85	81.55	0.30
56	83.5	82.17	1.33	83.58	82.49	1.09
57	85.21	83.1	2.11	85.30	83.42	1.88
58	86.62	84.02	2.6	86.70	84.32	2.38
59	88.01	84.91	3.1	88.11	85.21	2.91
60	89.42	85.79	3.63	89.49	86.08	3.41
61	90.82	86.64	4.18	90.85	86.93	3.92
62	92.2	87.48	4.72	92.19	87.76	4.43
63	93.82	88.41	5.41	93.84	88.68	5.16
64	95.42	89.32	6.1	95.47	89.59	5.89
65	97	90.21	6.79	97.09	90.47	6.62
66	98.59	91.07	7.52	98.68	91.33	7.36
67	100.18	91.92	8.26	100.28	92.16	8.12
68	101.76	92.74	9.02	101.87	92.98	8.89
69	103.35	93.54	9.81	103.47	93.78	9.69
70	104.91	94.32	10.59	105.05	94.55	10.50
71	106.53	95.05	11.48	106.67	95.30	11.37
72	108.15	95.82	12.33	108.31	96.03	12.27
73	109.75	96.59	13.16	109.92	96.80	13.12
74	111.34	97.35	13.99	111.51	97.55	13.96
75	112.95	98.08	14.87	113.14	98.28	14.87
76	114.53	98.79	15.74	114.76	98.98	15.78
77	116.11	99.48	16.63	116.35	99.66	16.69
78	117.67	100.14	17.53	117.88	100.31	17.57
79	120.14	101.18	18.96	120.31	101.33	18.98
80	122.57	102.18	20.39	122.70	102.31	20.39
81	125	103.13	21.87	125.12	103.23	21.89
82	127.38	104.03	23.35	127.51	104.10	23.41
83	129.73	104.89	24.84	129.86	104.92	24.94
84	132.09	105.7	26.39	132.22	105.71	26.51
85	134.52	106.48	28.04	134.65	106.47	28.19

Data Set 2

Axial Shortening calculated by COLECS				Axial Shortening calculated by ASCA		
Storey	Core Shortening	Column 2 Shortening	Differential Shortening	Core Shortening	Column 2 Shortening	Differential Shortening
	mm	mm	mm	mm	mm	mm
1	1.34	1.41	0.07	1.33	1.38	0.05
2	2.68	2.79	0.11	2.64	2.74	0.10
3	4.02	4.16	0.14	3.95	4.08	0.12
4	5.35	5.52	0.17	5.31	5.41	0.10
5	6.66	6.86	0.2	6.65	6.72	0.07
6	8.50	8.70	0.2	8.50	8.51	0.01
7	9.51	9.71	0.2	9.55	9.51	-0.05
8	10.52	10.71	0.19	10.60	10.49	-0.11
9	11.53	11.70	0.17	11.59	11.47	-0.13
10	12.54	12.68	0.14	12.63	12.43	-0.20
11	13.53	13.80	0.27	13.66	13.58	-0.08
12	14.52	14.92	0.4	14.68	14.72	0.04
13	15.51	16.02	0.51	15.68	15.84	0.16
14	16.48	17.13	0.65	16.68	16.92	0.24
15	17.91	18.32	0.41	18.02	18.14	0.11
16	19.34	19.49	0.15	19.37	19.34	-0.02
17	21.30	21.08	-0.22	21.32	20.88	-0.44
18	23.00	22.47	-0.53	22.92	22.28	-0.63
19	25.58	24.54	-1.04	25.38	24.38	-1.00
20	28.17	26.58	-1.59	28.10	26.46	-1.64
21	31.37	29.12	-2.25	31.25	29.01	-2.24
22	32.66	30.12	-2.54	32.48	30.04	-2.43
23	33.95	31.11	-2.84	33.69	31.05	-2.64
24	35.24	32.10	-3.14	34.92	32.05	-2.87
25	36.52	33.08	-3.44	36.14	33.05	-3.09
26	37.80	34.04	-3.76	37.36	34.04	-3.32
27	39.06	35.00	-4.06	38.56	35.01	-3.55
28	40.27	35.95	-4.32	39.80	35.97	-3.83
29	41.46	36.88	-4.58	41.01	36.92	-4.09
30	42.65	37.81	-4.84	42.23	37.87	-4.36
31	43.83	38.72	-5.11	43.43	38.80	-4.64
32	45.03	39.63	-5.4	44.66	39.71	-4.94
33	46.22	40.53	-5.69	45.87	40.62	-5.25
34	47.40	41.42	-5.98	47.07	41.53	-5.55
35	48.59	42.29	-6.3	48.30	42.42	-5.89
36	49.77	43.16	-6.61	49.52	43.30	-6.22
37	50.94	44.02	-6.92	50.73	44.17	-6.56
38	52.10	44.86	-7.24	51.92	45.03	-6.90
39	53.26	45.70	-7.56	53.11	45.87	-7.23
40	54.41	46.52	-7.89	54.28	46.71	-7.57
41	55.71	47.36	-8.35	55.60	47.56	-8.04
42	57.01	48.18	-8.83	56.90	48.39	-8.51
43	58.29	48.99	-9.3	58.20	49.21	-8.99

Axial Shortening calculated by COLECS				Axial Shortening calculated by ASCA		
Storey	Core Shortening	Column Shortening	Differential Shortening	Core Shortening	Column Shortening	Differential Shortening
	mm	mm	mm	mm	mm	mm
44	59.56	49.79	-9.77	59.48	50.02	-9.46
45	60.83	50.59	-10.24	60.75	50.82	-9.93
46	62.09	51.37	-10.72	62.03	51.61	-10.42
47	63.34	52.14	-11.2	63.29	52.39	-10.90
48	64.65	52.89	-11.76	64.62	53.15	-11.47
49	65.96	53.64	-12.32	65.93	53.90	-12.03
50	67.30	54.38	-12.92	67.30	54.64	-12.66
51	68.63	55.11	-13.52	68.66	55.37	-13.29
52	69.96	55.82	-14.14	70.01	56.09	-13.92
53	71.27	56.54	-14.73	71.34	56.81	-14.53
54	72.65	57.24	-15.41	72.75	57.52	-15.23
55	74.12	57.93	-16.19	74.26	58.21	-16.05
56	75.59	58.62	-16.97	75.76	58.90	-16.86
57	77.06	59.29	-17.77	77.24	59.57	-17.67
58	78.30	59.95	-18.35	78.48	60.23	-18.25
59	79.53	60.60	-18.93	79.70	60.88	-18.83
60	80.77	61.24	-19.53	80.96	61.51	-19.45
61	82.00	61.86	-20.14	82.20	62.14	-20.06
62	83.21	62.48	-20.73	83.42	62.75	-20.67
63	84.66	63.09	-21.57	84.86	63.35	-21.51
64	86.09	63.68	-22.41	86.28	63.94	-22.35
65	87.50	64.26	-23.24	87.69	64.51	-23.17
66	88.91	64.84	-24.07	89.10	65.08	-24.02
67	90.33	65.40	-24.93	90.51	65.63	-24.87
68	91.73	65.95	-25.78	91.90	66.17	-25.73
69	93.13	66.49	-26.64	93.29	66.70	-26.59
70	94.51	67.01	-27.5	94.67	67.22	-27.46
71	96.07	67.53	-28.54	96.20	67.72	-28.48
72	97.64	68.04	-29.6	97.75	68.21	-29.54
73	99.18	68.53	-30.65	99.28	68.69	-30.59
74	100.71	69.01	-31.7	100.79	69.16	-31.63
75	102.26	69.48	-32.78	102.31	69.61	-32.69
76	103.79	69.94	-33.85	103.83	70.06	-33.77
77	105.32	70.39	-34.93	105.33	70.49	-34.85
78	106.82	70.83	-35.99	106.81	70.90	-35.91
79	109.21	71.51	-37.7	109.13	71.55	-37.58
80	111.56	72.17	-39.39	111.41	72.16	-39.24
81	113.91	73.43	-40.48	113.67	73.35	-40.32
82	116.22	73.43	0.00	115.90	73.35	0.00
83	118.50	73.43	0.00	118.10	73.35	0.00
84	120.79	73.43	0.00	120.25	73.35	0.00
85	123.14	73.43	0.00	122.53	73.35	0.00

A5.4 Tributary area calculation

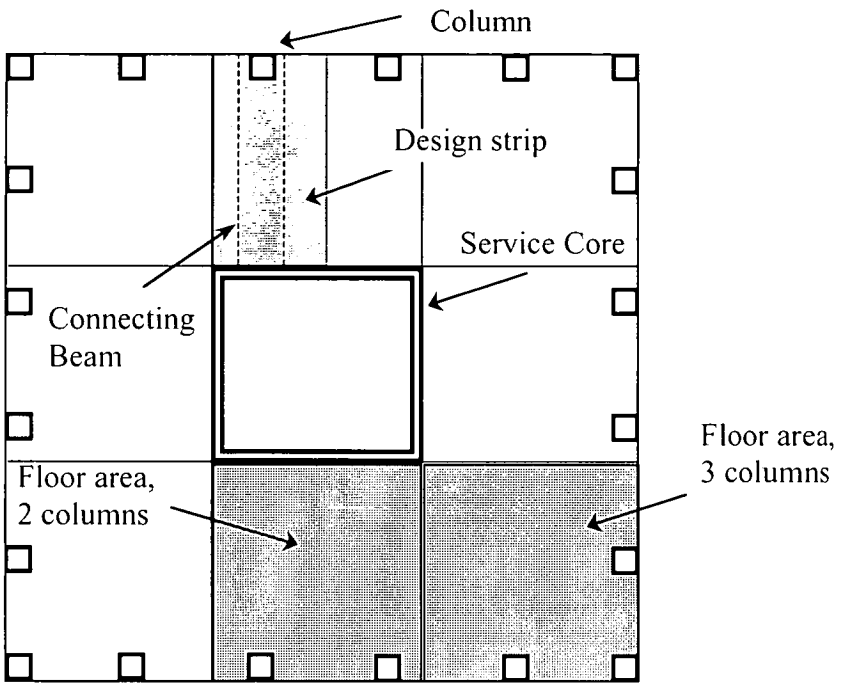


Figure A5.1 Floor plan showing design strip, column and core

The modelled floor area of the building in Figure A5.1 is broken up into eight equal parts. Assuming each part carries the same tributary loading then four of the eight parts are each supported by two columns whilst the other four parts are each supported by three columns. Therefore eight columns share half the floor load with the core and twelve columns share the other half. Thus the total floor load one column shares with the core is taken as the average in the following way

$$\text{shared floor area} = \frac{1/8 + 1/12}{2} = \frac{40/384}{2} = \frac{1}{19.2}$$

Thus the core effectively shares the total floor area with 19.2 columns each with equal load. Therefore 19.2 is used as the factor to increase the stiffness at the core connection of the framing member with one column. Importantly results did not change significantly if the factor is slightly larger or smaller. This is because the framing transfer onto the core is still quite small in comparison with the total load experienced. Figure 5.7 and 5.8 shows how little shortening in the core actually varies with framing.

A5.5 Percentage error difference between COLECS and ASCA

Data Set 1

Storey	CORE % ACCURACY	COL 1 % ACCURACY	Storey	CORE % ACCURACY	COL 1 % ACCURACY
1	99.5	98.2	44	100.0	99.4
2	99.8	97.3	45	100.0	99.4
3	99.8	97.1	46	100.0	99.5
4	99.7	97.2	47	99.9	99.5
5	99.8	97.1	48	99.9	99.5
6	99.9	97.1	49	99.9	99.5
7	100.0	97.2	50	99.9	99.5
8	100.0	97.3	51	99.9	99.5
9	100.0	97.4	52	100.0	99.5
10	100.0	98.8	53	100.0	99.6
11	100.0	97.6	54	100.0	99.6
12	99.9	97.7	55	100.0	99.6
13	99.9	97.9	56	99.9	99.6
14	99.9	98.1	57	99.9	99.6
15	99.9	98.2	58	99.9	99.6
16	99.9	98.4	59	99.9	99.7
17	100.0	98.6	60	99.9	99.7
18	99.9	98.7	61	100.0	99.7
19	100.0	98.9	62	100.0	99.7
20	99.9	99.0	63	100.0	99.7
21	100.0	99.4	64	99.9	99.7
22	100.0	99.5	65	99.9	99.7
23	100.0	99.7	66	99.9	99.7
24	100.0	99.8	67	99.9	99.7
25	100.0	99.9	68	99.9	99.7
26	100.0	100.0	69	99.9	99.7
27	100.0	99.9	70	99.9	99.8
28	99.9	99.9	71	99.9	99.7
29	99.9	100.0	72	99.9	99.8
30	99.9	100.0	73	99.8	99.8
31	99.9	100.0	74	99.8	99.8
32	99.8	100.0	75	99.8	99.8
33	99.8	100.0	76	99.8	99.8
34	99.8	100.0	77	99.8	99.8
35	99.8	99.9	78	99.8	99.8
36	99.9	99.8	79	99.9	99.9
37	99.9	99.7	80	99.9	99.9
38	99.9	99.7	81	99.9	99.9
39	100.0	99.6	82	99.9	99.9
40	100.0	99.6	83	99.9	100.0
41	100.0	99.5	84	99.9	100.0
42	100.0	99.5	85	99.9	100.0
43	100.0	99.5			

Data Set 2

Storey	CORE % ACCURACY	COL 2 % ACCURACY	Storey	CORE % ACCURACY	COL 2 % ACCURACY
1	99.0	97.7	44	99.9	99.5
2	98.4	98.1	45	99.9	99.5
3	98.4	98.1	46	99.9	99.5
4	99.2	98.0	47	99.9	99.5
5	99.8	98.0	48	99.9	99.5
6	100.0	97.8	49	100.0	99.5
7	99.6	97.9	50	100.0	99.5
8	99.3	98.0	51	100.0	99.5
9	99.4	98.0	52	99.9	99.5
10	99.3	98.1	53	99.9	99.5
11	99.1	98.4	54	99.9	99.5
12	98.9	98.6	55	99.8	99.5
13	98.9	98.9	56	99.8	99.5
14	98.8	98.8	57	99.8	99.5
15	99.4	99.0	58	99.8	99.5
16	99.9	99.2	59	99.8	99.5
17	99.9	99.1	60	99.8	99.6
18	99.6	99.2	61	99.8	99.6
19	99.2	99.3	62	99.7	99.6
20	99.8	99.5	63	99.8	99.6
21	99.6	99.6	64	99.8	99.6
22	99.4	99.7	65	99.8	99.6
23	99.2	99.8	66	99.8	99.6
24	99.1	99.9	67	99.8	99.6
25	99.0	99.9	68	99.8	99.7
26	98.8	100.0	69	99.8	99.7
27	98.7	100.0	70	99.8	99.7
28	98.8	99.9	71	99.9	99.7
29	98.9	99.9	72	99.9	99.7
30	99.0	99.9	73	99.9	99.8
31	99.1	99.8	74	99.9	99.8
32	99.2	99.8	75	100.0	99.8
33	99.2	99.8	76	100.0	99.8
34	99.3	99.7	77	100.0	99.9
35	99.4	99.7	78	100.0	99.9
36	99.5	99.7	79	99.9	99.9
37	99.6	99.7	80	99.9	100.0
38	99.7	99.6	81	99.8	99.9
39	99.7	99.6	82	99.7	99.9
40	99.8	99.6	83	99.7	99.9
41	99.8	99.6	84	99.6	99.9
42	99.8	99.6	85	99.5	99.9
43	99.8	99.5			

A5.6 Column and core framing from ASCA

Data Set 1

Column and core without framing action				Column and core with framing action		
Storey	Core Shortening	Column Shortening	Differential Shortening	Core Shortening	Column Shortening	Differential Shortening
	mm	mm	mm	mm	mm	mm
1	1.46	1.76	0.30	1.41	1.77	0.36
2	2.90	3.51	0.62	2.81	3.54	0.73
3	4.34	5.25	0.91	4.22	5.29	1.07
4	5.77	6.98	1.21	5.63	7.04	1.41
5	7.20	8.68	1.48	7.05	8.77	1.72
6	9.21	11.03	1.82	9.04	11.14	2.10
7	10.32	12.33	2.01	10.15	12.48	2.32
8	11.42	13.62	2.20	11.25	13.80	2.55
9	12.52	14.90	2.38	12.36	15.12	2.75
10	13.61	16.17	2.55	13.46	16.42	2.96
11	14.70	17.43	2.73	14.55	17.71	3.16
12	15.78	18.71	2.92	15.65	19.03	3.38
13	16.86	20.05	3.19	16.74	20.40	3.66
14	17.93	21.38	3.45	17.82	21.76	3.94
15	19.45	22.80	3.35	19.36	23.22	3.86
16	20.99	24.21	3.21	20.92	24.67	3.75
17	23.09	26.12	3.03	23.04	26.62	3.59
18	24.92	27.99	3.07	24.89	28.53	3.65
19	27.68	30.81	3.13	27.69	31.40	3.72
20	30.48	33.59	3.11	30.53	34.23	3.70
21	33.91	37.01	3.10	34.03	37.72	3.69
22	35.29	38.54	3.25	35.44	39.30	3.85
23	36.66	40.05	3.39	36.84	40.86	4.02
24	38.06	41.53	3.48	38.26	42.40	4.13
25	39.44	43.00	3.56	39.67	43.92	4.25
26	40.82	44.45	3.63	41.07	45.42	4.35
27	42.18	45.87	3.69	42.46	46.91	4.45
28	43.55	47.37	3.83	43.84	48.50	4.66
29	44.90	48.86	3.96	45.20	50.05	4.86
30	46.24	50.32	4.08	46.54	51.58	5.04
31	47.57	51.75	4.19	47.87	53.10	5.23
32	48.93	53.17	4.24	49.23	54.58	5.35
33	50.27	54.56	4.28	50.57	56.04	5.47
34	51.61	56.05	4.44	51.90	57.60	5.70
35	52.96	57.52	4.56	53.24	59.15	5.90
36	54.27	58.96	4.69	54.54	60.66	6.12
37	55.57	60.38	4.81	55.82	62.14	6.32
38	56.88	61.77	4.89	57.11	63.60	6.50
39	58.16	63.14	4.98	58.36	65.03	6.67
40	59.42	64.48	5.05	59.59	66.42	6.83
41	60.89	65.79	4.90	61.02	67.79	6.77
42	62.35	67.07	4.72	62.44	69.13	6.69
43	63.79	68.36	4.57	63.83	70.48	6.65

Column and core without framing action				Column and core with framing action		
Storey	Core Shortening	Column Shortening	Differential Shortening	Core Shortening	Column Shortening	Differential Shortening
	mm	mm	mm	mm	mm	mm
44	65.22	69.58	4.36	65.21	71.77	6.56
45	66.65	70.81	4.15	66.59	73.06	6.47
46	68.08	72.01	3.93	67.95	74.32	6.37
47	69.49	73.17	3.69	69.29	75.55	6.26
48	71.00	74.31	3.31	70.73	76.76	6.02
49	72.50	75.42	2.93	72.16	77.93	5.78
50	73.88	76.50	2.62	73.46	79.08	5.62
51	75.43	77.55	2.13	74.93	80.19	5.27
52	76.96	78.57	1.61	76.38	81.28	4.90
53	78.48	79.60	1.12	77.82	82.37	4.55
54	80.10	80.59	0.49	79.35	83.42	4.07
55	81.85	81.55	-0.30	81.01	84.45	3.44
56	83.58	82.49	-1.09	82.64	85.46	2.81
57	85.30	83.42	-1.88	84.27	86.44	2.17
58	86.70	84.32	-2.38	85.59	87.40	1.81
59	88.11	85.21	-2.91	86.93	88.35	1.42
60	89.49	86.08	-3.41	88.23	89.27	1.04
61	90.85	86.93	-3.92	89.51	90.18	0.66
62	92.19	87.76	-4.43	90.78	91.06	0.28
63	93.84	88.68	-5.16	92.34	92.04	-0.30
64	95.47	89.59	-5.89	93.87	92.99	-0.89
65	97.09	90.47	-6.62	95.39	93.91	-1.48
66	98.68	91.33	-7.36	96.89	94.82	-2.08
67	100.28	92.16	-8.12	98.40	95.70	-2.70
68	101.87	92.98	-8.89	99.89	96.55	-3.34
69	103.47	93.78	-9.69	101.40	97.38	-4.02
70	105.05	94.55	-10.50	102.90	98.19	-4.71
71	106.67	95.30	-11.37	104.43	98.97	-5.46
72	108.31	96.03	-12.27	105.98	99.72	-6.26
73	109.92	96.80	-13.12	107.51	100.51	-7.00
74	111.51	97.55	-13.96	109.03	101.28	-7.75
75	113.14	98.28	-14.87	110.59	102.02	-8.57
76	114.76	98.98	-15.78	112.13	102.73	-9.40
77	116.35	99.66	-16.69	113.65	103.42	-10.23
78	117.88	100.31	-17.57	115.11	104.08	-11.03
79	120.31	101.33	-18.98	117.45	105.09	-12.36
80	122.70	102.31	-20.39	119.76	106.07	-13.69
81	125.12	103.23	-21.89	122.12	107.00	-15.12
82	127.51	104.10	-23.41	124.46	107.89	-16.57
83	129.86	104.92	-24.94	126.78	108.73	-18.05
84	132.22	105.71	-26.51	129.12	109.55	-19.57
85	134.65	106.47	-28.19	131.55	110.34	-21.20

Data Set 2

Column and core without framing action				Column and core with framing action		
Storey	Core Shortening	Column Shortening	Differential Shortening	Core Shortening	Column Shortening	Differential Shortening
	mm	mm	mm	mm	mm	mm
1	1.34	1.41	0.07	1.32	1.59	0.26
2	2.68	2.79	0.11	2.65	3.15	0.50
3	4.02	4.16	0.14	3.97	4.70	0.72
4	5.35	5.52	0.17	5.29	6.23	0.95
5	6.66	6.86	0.20	6.58	7.75	1.17
6	8.50	8.70	0.20	8.40	9.84	1.44
7	9.51	9.71	0.20	9.39	10.99	1.59
8	10.52	10.71	0.19	10.39	12.13	1.73
9	11.53	11.70	0.17	11.39	13.25	1.86
10	12.54	12.68	0.14	12.39	14.37	1.99
11	13.53	13.80	0.27	13.36	15.64	2.28
12	14.52	14.92	0.40	14.34	16.91	2.57
13	15.51	16.02	0.51	15.32	18.16	2.84
14	16.48	17.13	0.65	16.28	19.42	3.14
15	17.91	18.32	0.41	17.69	20.77	3.08
16	19.34	19.49	0.15	19.10	22.10	3.00
17	21.30	21.08	-0.22	21.04	23.92	2.88
18	23.00	22.47	-0.53	22.71	25.51	2.79
19	25.58	24.54	-1.04	25.26	27.88	2.62
20	28.17	26.58	-1.59	27.82	30.22	2.40
21	31.37	29.12	-2.25	30.97	33.13	2.16
22	32.66	30.12	-2.54	32.25	34.28	2.04
23	33.95	31.11	-2.84	33.52	35.43	1.91
24	35.24	32.10	-3.14	34.79	36.57	1.78
25	36.52	33.08	-3.44	36.06	37.70	1.64
26	37.80	34.04	-3.76	37.32	38.81	1.49
27	39.06	35.00	-4.06	38.56	39.92	1.36
28	40.27	35.95	-4.32	39.76	41.02	1.27
29	41.46	36.88	-4.58	40.93	42.10	1.17
30	42.65	37.81	-4.84	42.10	43.18	1.08
31	43.83	38.72	-5.11	43.27	44.24	0.97
32	45.03	39.63	-5.40	44.45	45.30	0.85
33	46.22	40.53	-5.69	45.63	46.35	0.73
34	47.40	41.42	-5.98	46.79	47.39	0.60
35	48.59	42.29	-6.30	47.96	48.41	0.44
36	49.77	43.16	-6.61	49.13	49.42	0.30
37	50.94	44.02	-6.92	50.28	50.43	0.15
38	52.10	44.86	-7.24	51.43	51.42	-0.01
39	53.26	45.70	-7.56	52.57	52.40	-0.17
40	54.41	46.52	-7.89	53.70	53.37	-0.34
41	55.71	47.36	-8.35	54.99	54.35	-0.63
42	57.01	48.18	-8.83	56.27	55.32	-0.95
43	58.29	48.99	-9.30	57.53	56.27	-1.26

Column and core without framing action				Column and core with framing action		
Storey	Core Shortening	Column Shortening	Differential Shortening	Core Shortening	Column Shortening	Differential Shortening
	mm	mm	mm	mm	mm	mm
44	59.56	49.79	-9.77	58.78	57.21	-1.57
45	60.83	50.59	-10.24	60.03	58.15	-1.88
46	62.09	51.37	-10.72	61.28	59.07	-2.21
47	63.34	52.14	-11.20	62.51	59.97	-2.54
48	64.65	52.89	-11.76	63.80	60.86	-2.94
49	65.96	53.64	-12.32	65.09	61.74	-3.35
50	67.30	54.38	-12.92	66.42	62.61	-3.81
51	68.63	55.11	-13.52	67.73	63.47	-4.26
52	69.96	55.82	-14.14	69.04	64.30	-4.74
53	71.27	56.54	-14.73	70.33	65.15	-5.18
54	72.65	57.24	-15.41	71.70	65.97	-5.72
55	74.12	57.93	-16.19	73.15	66.78	-6.37
56	75.59	58.62	-16.97	74.60	67.59	-7.01
57	77.06	59.29	-17.77	76.05	68.38	-7.68
58	78.30	59.95	-18.35	77.28	69.15	-8.13
59	79.53	60.60	-18.93	78.49	69.91	-8.58
60	80.77	61.24	-19.53	79.72	70.66	-9.06
61	82.00	61.86	-20.14	80.93	71.38	-9.55
62	83.21	62.48	-20.73	82.13	72.10	-10.03
63	84.66	63.09	-21.57	83.56	72.81	-10.75
64	86.09	63.68	-22.41	84.97	73.49	-11.48
65	87.50	64.26	-23.24	86.36	74.16	-12.20
66	88.91	64.84	-24.07	87.76	74.83	-12.93
67	90.33	65.40	-24.93	89.16	75.48	-13.69
68	91.73	65.95	-25.78	90.55	76.11	-14.44
69	93.13	66.49	-26.64	91.93	76.72	-15.21
70	94.51	67.01	-27.50	93.30	77.31	-15.99
71	96.07	67.53	-28.54	94.84	77.90	-16.94
72	97.64	68.04	-29.60	96.40	78.47	-17.93
73	99.18	68.53	-30.65	97.93	79.02	-18.91
74	100.71	69.01	-31.70	99.45	79.56	-19.89
75	102.26	69.48	-32.78	100.99	80.07	-20.91
76	103.79	69.94	-33.85	102.51	80.58	-21.93
77	105.32	70.39	-34.93	104.03	81.07	-22.96
78	106.82	70.83	-35.99	105.52	81.54	-23.98
79	109.21	71.51	-37.70	107.90	82.27	-25.64
80	111.56	72.17	-39.39	110.25	82.96	-27.29
81	113.91	73.43	-40.48	112.59	84.26	-28.32
82	116.22	74.88	-41.34	114.89	86.03	-28.86
83	118.50	76.33	-42.17	117.17	87.77	-29.40
84	120.79	77.78	-43.01	119.46	89.49	-29.97
85	123.14	79.23	-43.91	121.81	91.18	-30.63

Chapter 6

SOFTWARE DEVELOPMENT

6.1 Introduction

Chapter 4 details the essential procedure required to calculate axial deformations for a simple two pair column stack of two storeys. TCBs consisting of more pairs and upwards of 100 storeys require levels of calculation prohibitive by hand. One of the objectives of this research was to enable axial deformations to be calculated for different structures and increase the simplicity in interpreting results. For this a computer program was required. Before beginning to write code, a good understanding of the task and the objectives of the project need to be specified. The objectives are based on the desired application of the program within the scope of the project.

In order to understand the task and create a software package that fulfils the objectives, a number of defined processes needed to be implemented;

i) develop a model for calculating axial deformations.
ii) isolate the parameters of the model
iii) create a streamlined approach to link all parameters
iv) develop code

Process (i) is discussed in detail in the previous chapters. Processes (ii), (iii) and (iv) are used to develop the final software package. Of these three processes the language used to develop the code dictates the approach taken towards process (ii) and (iii). A sequential programming language that relies on procedures to link global variables, such as Fortran or Pascal, drastically differs from an object orientated program language like C++ or Java. Due to the fact that a multi-storey building is made up of a number of components, for example, columns, beams and wall, an object orientated language was chosen to develop

the program. Java is a web based object orientated language with a vast library of classes available for download from the world wide web. It is also well supported at the University of Tasmania so Java was chosen as the programming language.

An object orientated programming language consists of breaking up the model into smaller objects (Eckstein 1998). Each object can be a physical parameter of the model or a linking object. In this case the aim is to model a collection of vertical elements that each interact and react according to their own specific make up. For this reason the project is well suited to object orientated programming and we need to isolate the parameters accordingly.

6.2 Parameters of the Model

The model becomes more complex as more parameters are introduced. Conversely, if too few parameters are used, the model becomes less accurate or less versatile (Deitel 1998). It is necessary to find a balance between complexity and accuracy. This is done by selecting the right parameters of the program.

In the case of a TCB, the following major five parameters are sufficient to enable the program to be structured and still be accurate;

i) building environment
ii) physical structure
iii) materials
iv) building cycle
v) loading procedure

The five parameters contain additional sub parameters as follows;

i) building environment
   location , climatic conditions

ii) physical structure
   column, core, wall and beam elements

iii) materials
   concrete, steel, material shape (section)

iv) building cycle
   stages of construction of columns , cores, beams, introduction of framing

v) loading procedure
   in parallel with the building cycle, includes dead, live and construction loads

By choosing an object orientated programming language, parameters can be created separately as classes. The class is essentially an object that contains a set of variables that can be turned on and off, or given specific values or tasks at different times during implementation of the program (Deitel 1998). This is critical because it helps to define the underlying *dimension* of the program. Of the five major parameters, the building environment, the physical structure and the materials are fixed and can be created and given values at any time. The building cycle is time dependent and the loading procedure can be linked to the building cycle.

As object orientated programming languages allow for classes to be created at any time and manipulated as required (Deitel 1998), the building cycle complicates the program by dictating when something happens along a predetermined cycle. In essence it defeats the power of creating classes that can be manipulated at any point in time. Thus the building cycle can be isolated as a duplicated parameter as we can simply specify time measured in days as a *dimension* for setting up the construction cycle and allow users to generate their

own construction times. The loading parameter is then linked to time rather than the building cycle. This introduces the possibility of time related errors which simply need to be controlled at the input stage.

By removing the building cycle parameter (iv) four main classes can be defined;

i) environment
ii) structural element
iii) material
iv) loading element

The structural element, material and loading element classes are all contained within the building environment. Each of these three classes consists of sub-classes with each sub-class containing separate parameters and links.

The main sub-classes of each main class are;

i) structural element
   a) column (including a core)
   b) beam
   c) wall

ii) material
   a) section property
   b) steel property
   c) concrete property

iii) loading
   a) dead load
   b) live loads

In addition to these main classes, a framing class, coordinate class, concrete model classes and numerous dialog and calculation classes are required. The full class structure can be viewed in Appendix 6.1.

6.2.1 Linking classes

Classes are linked by how and when they need to communicate between one another. Each class is a separate piece of code that performs like its own program and interacts by accepting or passing values to other variables in other classes at different points in time. On account of this, it is easy to write classes, like a concrete model class, and test them thoroughly as separate entities and simply attach them to the package that forms the program. The best way to see how classes interact is via flow charts. The two flow charts in Section 6.2.2 below run through the input procedure of the program and the basic internal working of the program.

6.2.2 Flow chart

The following flow chart in Figure 6.1 illustrates the processes required to set up the data required to perform axial shortening calculations. Figure 6.2 demonstrates the logic process the program performs to arrive at axial shortening values. Links between boxes represent the only available paths.

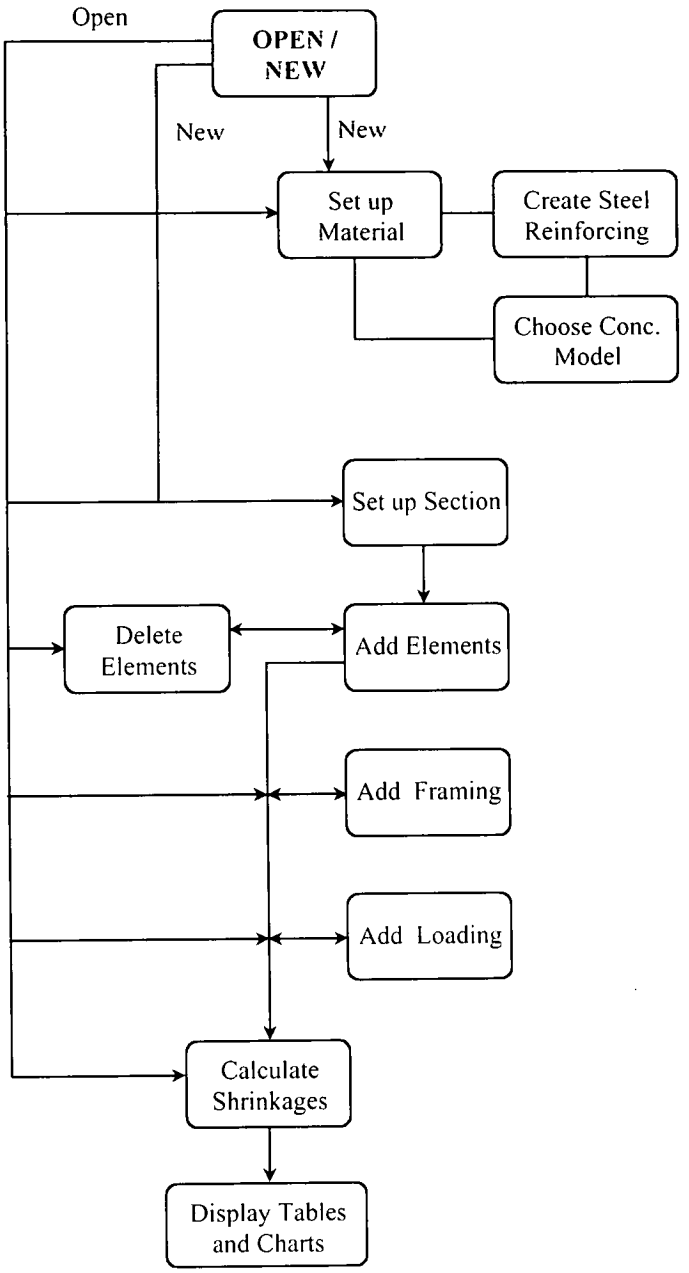


Figure 6.1 Flow chart to set up the building model

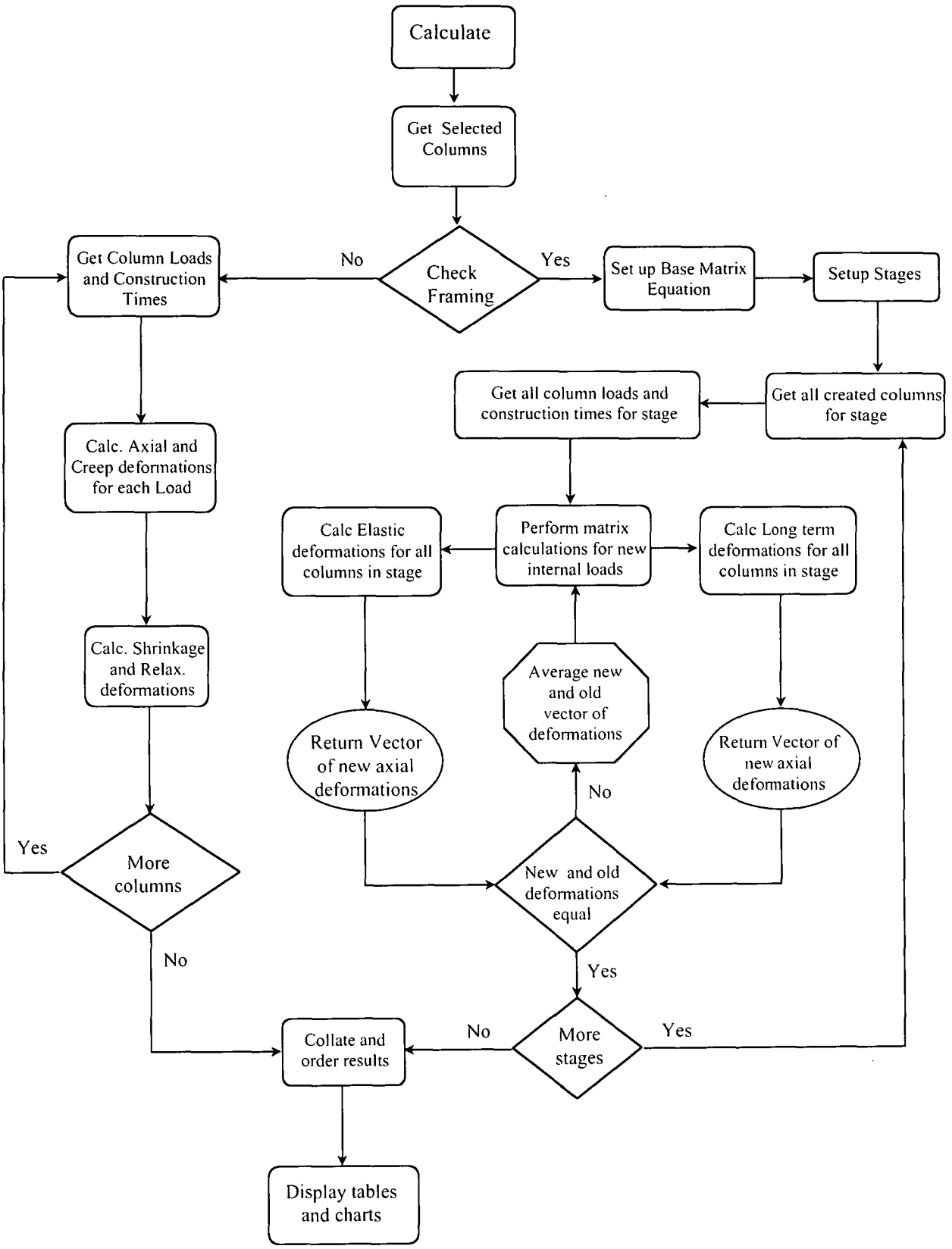


Figure 6.2 Flow chart of logic structure for performing calculations in the program

6.3 Developing the Code

Two aspects of the software needed to be coded. The first aspect was the generation of classes, number crunching and manipulation of classes. The other aspect was the creation of the graphical user interface for inputting and outputting data (Knudsen 1999). Java contains a number of components that make it possible to create a graphical input interface with outputs in table form (Jardin 1997). The sophistication of the input and output interface is low as the main aim is to produce a working program, not a package for resale.

The user interface consists of menus that creates dialog boxes. Inputs are prompted and any errors highlighted at time of entry. Entering information can be performed in any order[1], however, the most logical way is to follow the steps at each menu from left to right, top to bottom.

Section 2 of the thesis is dedicated to the program and contains instructions on how the program works as well as its limitations and the source code. Refer Section 2 for more information on the way the program works

6.4 Summary to Chapter

The objectives and the process for setting up the model to enable programming of the code are discussed in this chapter. From this, the relevant variables of the model are selected with emphasis on keeping the program as simple as possible but still enabling the program to be versatile. The selection of the coding language and the reasons for such a selection are outlined. How the program works to achieve the desired output is also presented with emphasis on the fact that the program results need to be calibrated in some way to provide realistic accurate results. The way the program is designed and how the user operates the program is explained.

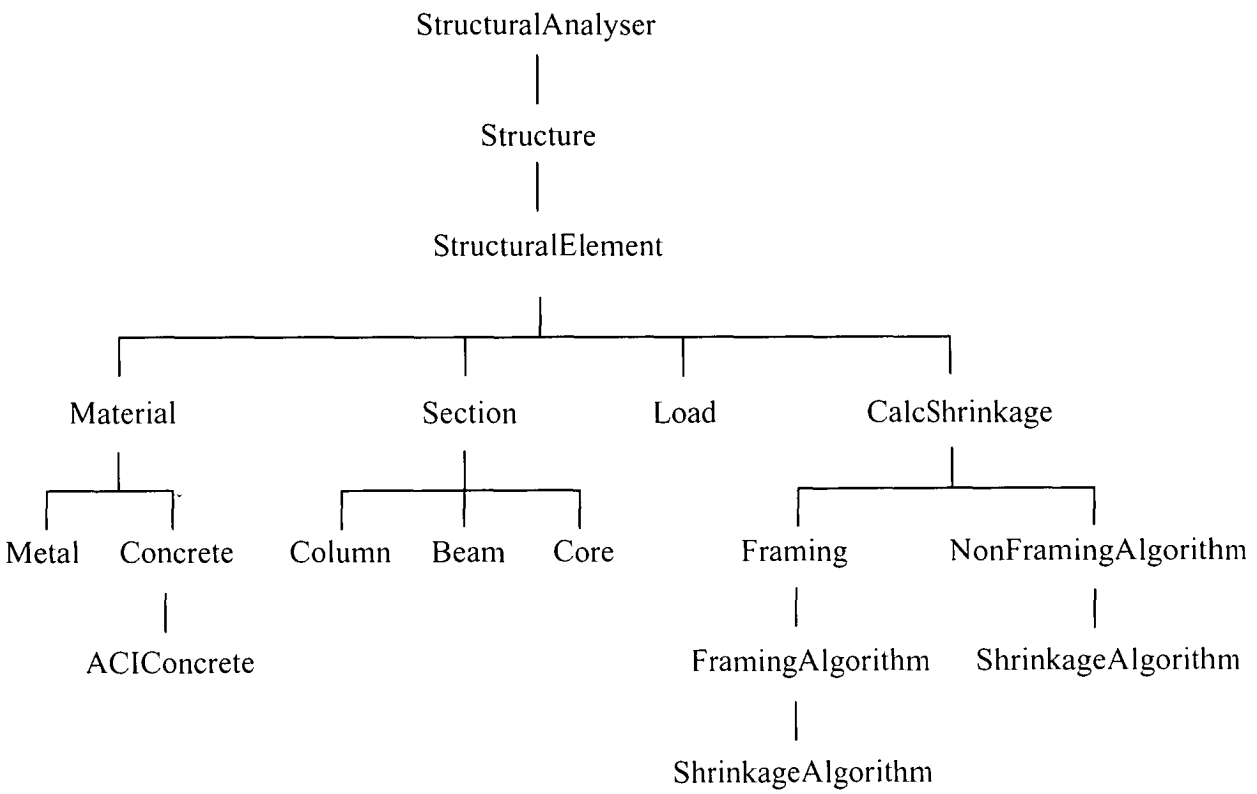
---

[1] The program will only allow some inputs after other related information is entered

Appendix

A 6.1 Class structures used in ASCA

i) Building Model



ii) Accessor Classes

AddFraming

EditElement

NewElement

NewMaterial

NewSection

iii) GUI Classes

AnalyserFrame

TableDialog

JScrollPaneAdjuster

JTableRowHeaderResizer

LoadsDialog

iv) Error Classes

intComparator

InvalidLoadException

InvalidParameterException

InvalidPositionException

dblComparator

Assert

v) Other Classes

Coordinate

Rounding

Chapter 7

CONCLUSION

7.1 Introduction

TCBs suffer from axial shortening in all vertical load-bearing elements. Calculation of axial shortening for a single column requires complex second-order analysis. The introduction of framing action between multiple columns greatly increases the complexity of the analysis required. An ordered approach for analysing the mechanics of axial shortening for a framed structure is required to ensure the process of axial shortening can be modelled accurately.

Previous research and proposed methods for dealing with axial shortening are considered in detail so that a model could be generated that would enable simpler more accurate analysis of axial shortening to be made, with the addition of solving the added complexity introduced when framing action is considered.

The stages that led to the development of the model are;

i) development of a model for a concrete column.
ii) development of a method to account for framing
iii) applying this method to a TCB under construction

With a model developed that enabled axial shortening of a framed building to be calculated the author produced a computer package that performed all the calculations required when solving axial shortening for a TCB with framing between columns and cores. Data from an 85 storey building was considered so the model could be calibrated and the effect that framing had on axial shortening could be investigated.

7.2 Development of Concrete Column Models

A reinforced concrete model was developed using previous research from Bazant (1995), Koutsoukis (1995), Beasley (1987) and the ACI Code (1986). The model needed to contain all the parameters necessary to determine axial shortening and be able to be applied to the model used to account for framing action as well as the building cycle for a framed TCB.

The model centred around the total strain equation below

$$\varepsilon(t) = \frac{\sigma_c(s)}{E'(t,s)} + \varepsilon_{sh}(t) + \frac{\Delta\sigma(t)}{E''(t,s)} \tag{7.1}$$

By considering a reinforced concrete section, four fundamental components of axial shortening were derived. These were derived in the form below as this was the best way to incorporate the four components into a framed TCB

i) elastic shortening

$$\delta_e(s) = \frac{P'(s)\,L}{A_c(1+n(s)\,p)E_c(s)} \tag{7.2}$$

ii) creep shortening

$$\delta_c(t,s) = \frac{P'(s)L}{A_c(1+n(s)\,p)}\left[\frac{1}{E'(t,s)} + \frac{p\,(n(s)-n'(t,s))}{(1+n''(t,s)\,p)\,E''(t,s)}\right] - \frac{P'(s)\,L}{A_c(1+n(s)\,p)\,E_c(s)} \tag{7.3}$$

iii) shrinkage shortening

$$\delta_s(t) = \varepsilon_{sh}(t).L \tag{7.4}$$

iv) reinforcing stiffness

$$\delta_r(t) = \frac{\varepsilon_{sh}(t)\,E_s\,p\,L}{(1+n''(t,t_0)\,p)\,E''(t,t_0)} \tag{7.5}$$

By considering a possible building cycle and the principle of superposition, a complete approach is developed that is directly applicable to a TCB with framing action. The total four components can be simply expressed by (7.6)

$$\delta_{tot}(s_n) = \sum_{i=1}^{n}\delta_e(s_i) + \sum_{i=1}^{n}\delta_c(s_{n,n-1}) + \sum_{i=1}^{n}\delta_s(s_n) - \sum_{i=1}^{n}\delta_r(s_n) \qquad (7.6)$$

Each of the four components above need to be addressed in a different manner when framing action is considered. Refer Figure 7.1 below. The elastic component was the easiest to model as framing forces and axial shortening occur concurrently. Shrinkage and reinforcing stiffness were both unrelated to load and were only a function of time and differential shortening, so incorporating framing action was also relatively straight-forward. Creep shortening was a function of all parameters so it needed to be considered at every stage of the model. A segmented model for creep at each stage in a building cycle was developed for each column. At the appropriate time the creep shortening was applied to the framing model.

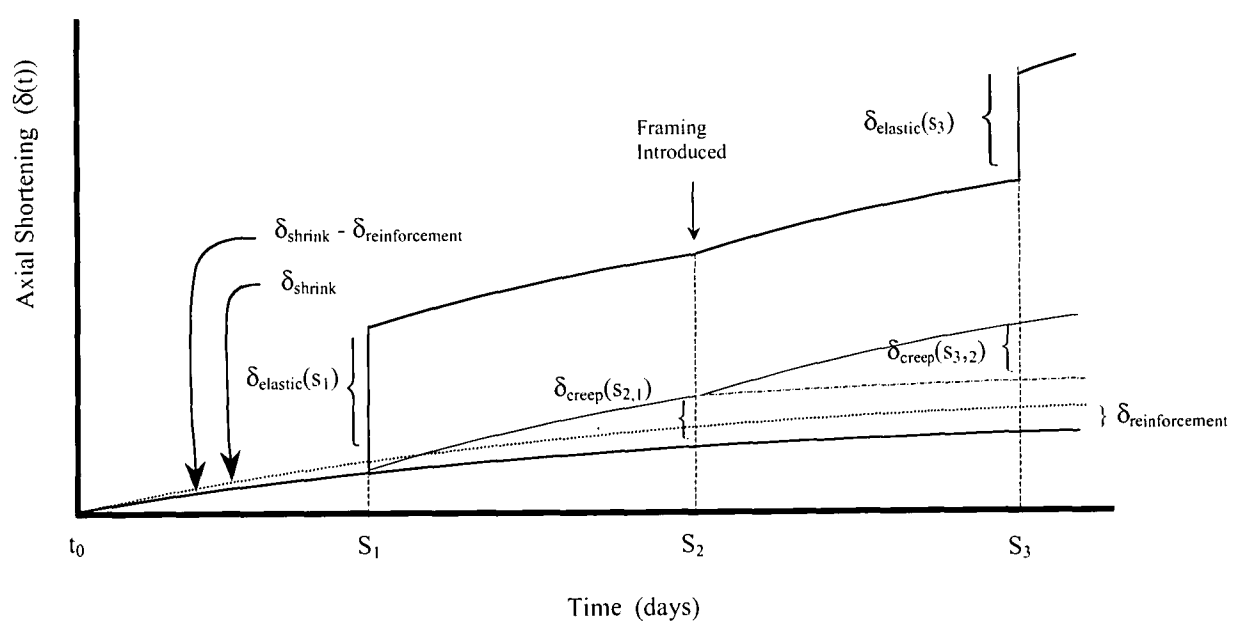


Figure 7.1 Pictorial graph of axial shortening with framing effects introduced at (s_2)

7.3 Development of Framing Model

The method for handling framing action was based around a stiffness matrix. A force-deflection relationship that could be set up in matrix form provided a way of managing force transfer associated with framing action and axial shortening. The models developed to calculate axial shortening components provided a direct relationship between internal force and deflection. In order to simplify the matrix and make it more specific to the problem at hand, the framed column pair for TCBs proposed by Warner (1975) was adapted to multiple storeys. By considering the column pair and all the forces at each node the matrix equation below was developed;

$$\begin{matrix} 1^{st} \\ \text{Storey} \\ \\ 2^{nd} \\ \text{Storey} \end{matrix}\left[\begin{array}{cc:cc} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ \hdashline & & & \end{array}\right]\begin{bmatrix} P^{I}_{1,1} \\ P^{I}_{1,2} \\ \hdashline P^{I}_{2,1} \\ P^{I}_{2,2} \end{bmatrix} = \begin{bmatrix} P^{E}_{1,1} \\ P^{E}_{1,2} \\ \hdashline P^{E}_{2,1} \\ P^{E}_{2,2} \end{bmatrix} + \left[\begin{array}{cc:cc} -k_{1,1/2} & k_{1,1/2} & 0 & 0 \\ k_{1,2/1} & -k_{1,2/1} & 0 & 0 \\ \hdashline & & & \end{array}\right]\begin{bmatrix} \delta_{1,1} \\ \delta_{1,2} \\ \hdashline \delta_{2,1} \\ \delta_{2,2} \end{bmatrix}$$

Matrix equation 3.1

The equation was expanded to cover any number of columns and any number of storeys. This enabled a structure to be created with multiple framing of all members on each level of the building. Axial shortening of every column on every storey could then be determined at any period after construction.

$$\begin{bmatrix} [A]_m & [-A]_m & & & & & & & \\ & [A]_m & [..]_m & & & & & & \\ & & [..]_m & [..]_m & & & & & \\ & & & [A]_m & [-A]_m & & & & \\ & & & & [A]_m & [..]_m & & & \\ & & & & & [..]_m & [..]_m & & \\ & & & & & & [A]_m & & \end{bmatrix}\begin{Bmatrix} \{P^{I}_1\}_m \\ \{P^{I}_2\}_m \\ \dots \\ \{P^{I}_j\}_m \\ \{P^{I}_{j+1}\}_m \\ \dots \\ \{P^{I}_n\}_m \end{Bmatrix} = \begin{Bmatrix} \{P^{E}_1\}_m \\ \{P^{E}_2\}_m \\ \dots \\ \{P^{E}_j\}_m \\ \{P^{E}_{j+1}\}_m \\ \dots \\ \{P^{E}_n\}_m \end{Bmatrix} + \begin{bmatrix} [k_1]_m & & & & & & \\ & [k_2]_m & & & & & \\ & & [...]_m & & & & \\ & & & [k_j]_m & & & \\ & & & & [k_{j+1}]_m & & \\ & & & & & [...]_m & \\ & & & & & & [k_n]_m \end{bmatrix}\begin{Bmatrix} \{\delta\delta_1\}_m \\ \{\delta\delta_2\}_m \\ \dots \\ \{\delta\delta_j\}_m \\ \{\delta\delta_{j+1}\}_m \\ \dots \\ \{\delta\delta_n\}_m \end{Bmatrix}$$

General matrix equation 3.6

From this base matrix solution, the building cycle was introduced by highlighting all the possible stages associated with the construction of the building. At each unique stage the base matrix was developed further to reflect the actual shortening actions present between the unique stages. This then enabled a full analysis of axial shortening to be made for any particular structure, adopting its own specific construction process. The associated effect of creep shortening is also accounted for at these stages.

The method developed provides differential axial shortening for columns and cores at any chosen day from the commencement of construction, even at periods during construction.

7.4 Software

The model developed was used to produce a software package ASCA. The programming language chosen was Java as it was an object oriented language with good libraries[1]. The software has a basic graphic user interface and produces results in a table form. An Excel® spreadsheet has been created to produce graphical outputs for the generated results. The program is based on the ACI (1986) model for creep and shrinkage coefficients, although it can be adapted to consider other creep and shrinkage models. Framing action can be removed so that isolated axial shortening analysis can be made. This allowed results from the program to be extensively calibrated against COLECS (1987). Both programs produced very similar results for two sets of data from the Bayoke Tower II building in Thailand.

Figure 7.2 compares actual differences between COLECS and ASCA for non framed axial deformations for the two 85 storey column pairs of the Bayoke Tower II building. The majority of results calculated by both programs varied by less than 0.5% with a maximum difference of 3.8% for the entire set of data.

[1] Libraries refer to pre-coded methods available to programmers to assist with repetitive, common or mathematical code when writing a program

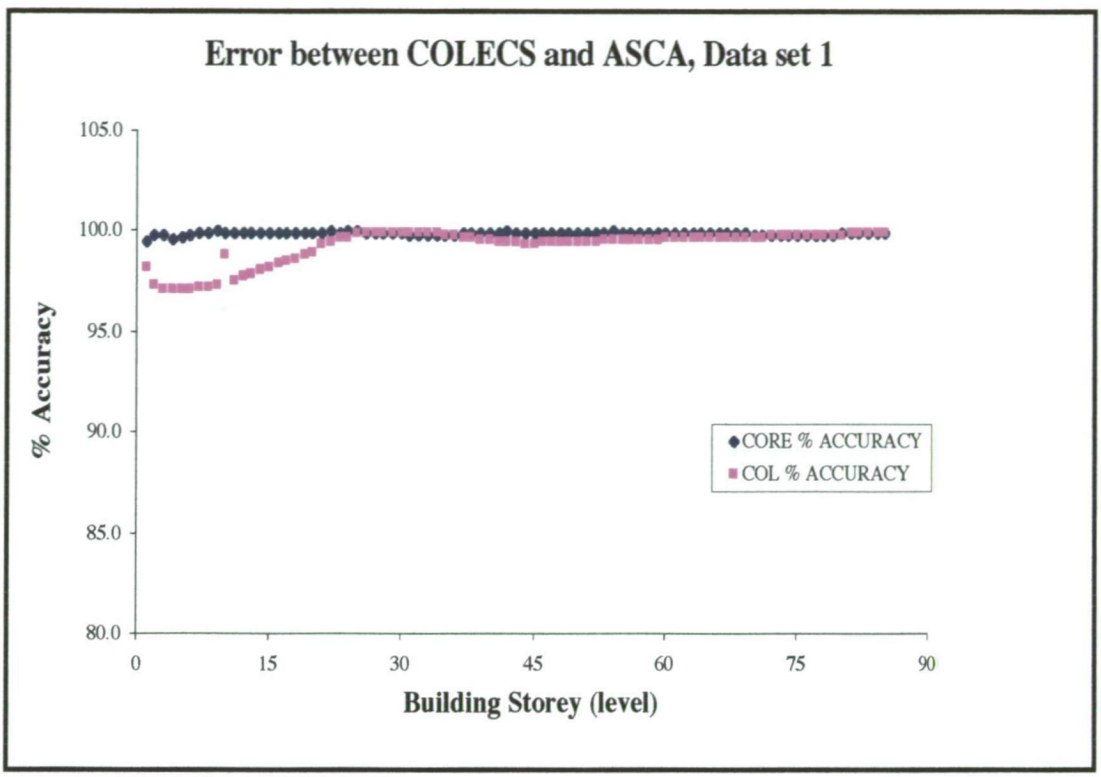


Figure 7.2 Percentage error between ASCA and COLECS for Data Set 1

7.5 Results

The two sets of data used to calibrate ASCA with COLECS were then re-calculated by ASCA with framing action introduced. A connecting reinforced beam was modelled with a full moment connection between the column and core for each set of data. Refer Figure 5.6. Attempts were made to model the connecting beam between the column and the core, considering a factored tributary floor area for the core. Importantly results did not appear to

change significantly if the assumed floor area for the core was slightly larger or smaller. Although no definitive study has been conducted, it is believed that this is because the framing transfer onto the core is still quite small in comparison with the total load experienced and it is in fact the column that experiences the greatest effect of framing action. Refer figure 7.3 and 7.4 below as taken from Figure 5.7 and 5.8 in Chapter 5.

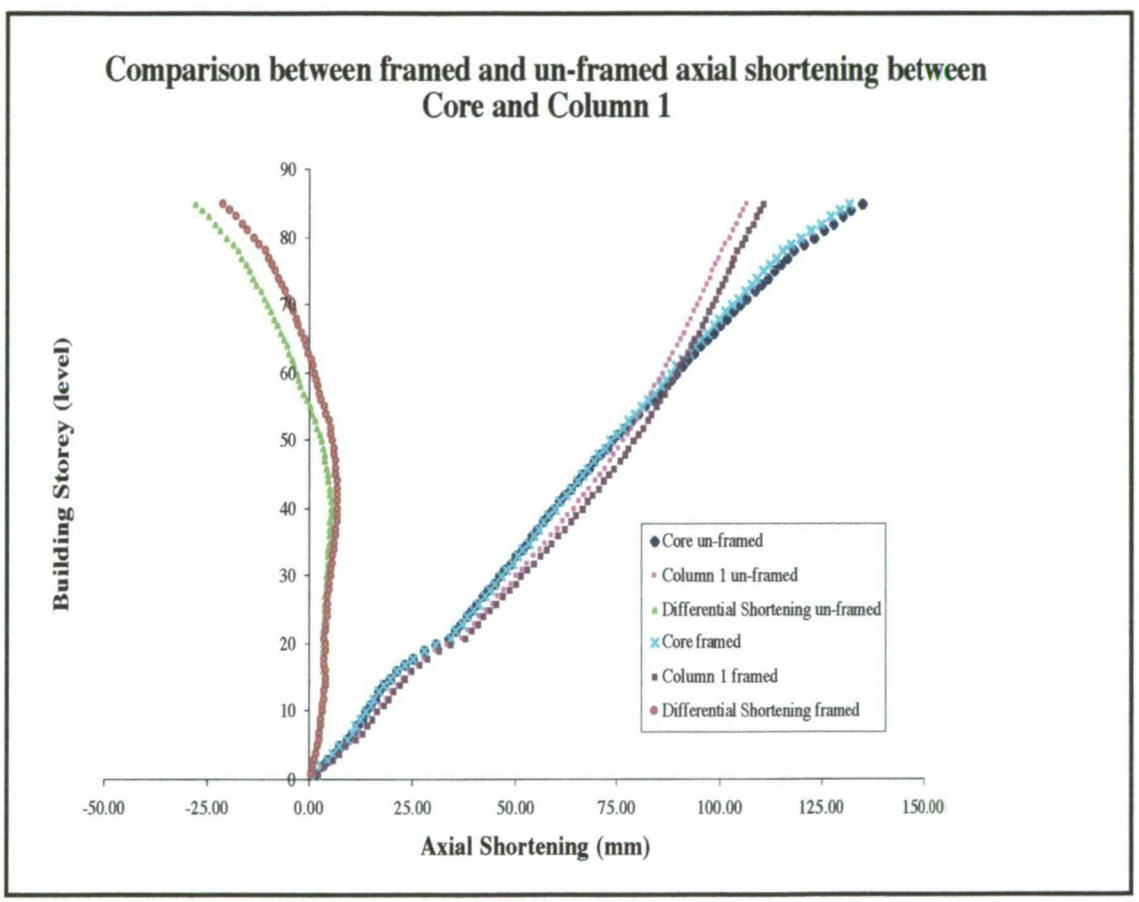


Figure 7.3 Chart of axial shortening for Data Set 1

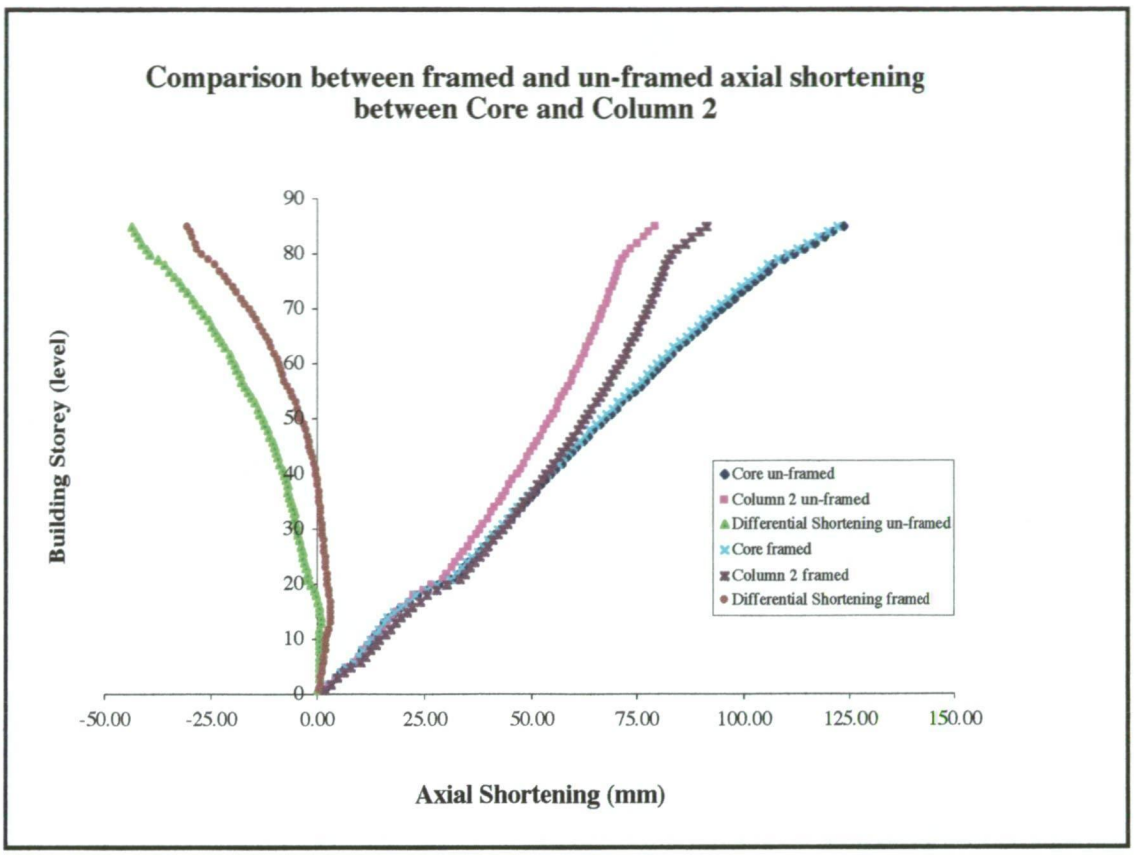


Figure 7.4 Chart of axial shortening for Data Set 2

The two data sets studied were on a pair of columns in the Bayoke Tower II with Data Set 1 representing the core and an exterior column, and Data Set 2 representing the core and a corner column.

The total differential shortening at the 85th storey between the column and core for Data Set 1 was 28.19 mm for an unframed analysis and 21.2 mm for a framed analysis, representing a 24.8% decrease in actual differential axial shortening when framing was introduced. Refer Fig 5.9 in Chapter 5 for differences.

The total differential shortening at the 85th storey between the column and core for Data Set 2 was 43.91 mm for an unframed analysis and 30.63 mm for a framed analysis, representing

a 30.2% decrease in actual differential axial shortening when framing was introduced. Refer Fig 5.10 in Chapter 5 for differences.

By considering these results as a function of total axial deformations of the core, the difference in shortening of the column for Data Set 1 is 20.1% (unframed) and 16% (framed), while for Data Set 2 the difference in shortening of the column is 35.65% of the core (unframed) and 25% (framed). For Data Set 1 this represents only a 4% reduction as a function of total axial shortening when framing is introduced, where as in Data Set 2 there is a 10.65% reduction. The reason for the difference between the two data sets is explained further in conclusion point ii0 below. Figure 5.12 and 5.13 in Chapter 5 show percentage differences for each level for Data Set 1 and Data Set 2 respectively.

Each data set from the Bayoke Tower II produced different differential shortening results even though they were from the same building with the same construction cycle. From This study, by considering these results, three important conclusions can made ;

i) Framing action reduces the differential axial shortening between two elements
ii) The core generally experiences less change to axial shortening than the column
iii) The effect of framing cannot be estimated by a relative percentage adjustment applied uniformly to each storey.

i) At all storeys in both sets of data the differential shortening between the column and the core was reduced. This is due to the load re-distribution via shear force transfer. It is obvious from the two sets of data analysed that framing had the biggest effect when differential axial shortening was the largest. Refer Figure 7.3 and 7.4.

ii) Figure 7.3 and 7.4 also show that in both sets of data the core experienced little change after framing was considered. This can be attributed to the level of redistribution of forces. The core experiences substantially more load than the column so the percentage of load distributed by framing is significantly less at the core than the column. In addition the core

is generally much stiffer than the column thus the changes in axial forces have less effect on the core.

In most cases the core (which carries the greatest proportion of load) experiences more axial shortening then the peripheral columns. Refer Figure 7.3 and 7.4. Because of this framing action causes an increase in axial force in the column. This is an important aspect of framing that needs to be considered by designers when calculating column forces. Stiffening the column has the potential to further increase differential axial shortening, thus an even greater transfer of shear forces to the column will occur when framing is considered.

iii) In some cases such as Data Set 1, at lower storeys the column has a greater total axial shortening than the core and at higher storeys the core has greater total axial shortening than the column. Refer Figure 7.4. This arrangement results in the column experiencing a reduction in internal axial force at lower levels and an increase in internal axial force a higher levels. Figure 5.12 in Chapter 5 demonstrates the effect this has on differential axial shortening. This can be compared tho Figure 5.13 in Chapter 5 for Data Set 2 where the core always has a greater total axial shortening than the column.

An important conclusion to be drawn from this result is that the actual effect framing has on axial shortening in not consistent across the entire structure. The results from Data Set 2 could be used to support a theory proposed that framing effects can be adjusted evenly by simply considering the residual stiffness of a similar frame (Fintel 1986). In Data Set 2 axial shortening across the structure is relatively even. However Data Set 1 produces a varied alteration range from 2% to 8%. It appears from these two data sets that the effect of framing is far more specific to the loading patterns and the individual relationships between each element at each floor. Designers need to be aware of how each column pair is effected by framing at each storey. In particular differential column shortening with the core needs to be extensively analysed as it is more likely that the column will experience an increase in internal axial force due to framing. The capacity of the column needs to be checked for such increases.

7.6 Future Work

This research has concentrated on determining a model that will allow axial shortening to be calculated for a TCB with framing action. The actual method developed to account for framing action is considered more relevant than the method adopted to determine creep and shrinkage coefficients (ACI 1986). While a working method has been achieved, greater versatility and accuracy can be achieved by further work on the areas listed below;

i) The work in this research provides a workable solution based on segmenting the creep an element experiences, so that at the relevant stages creep effects can be considered and then, by the principle of superposition, summed together to determine the total creep shortening. Further work on the effect that framing has on creep is recommended, especially in the area of load removal and segmenting creep into smaller parts.

ii) Incorporate other creep and shrinkage models in to the program. These could include the B3 model proposed by Bazant (1995) or the new AS3600 model proposed by Gilbert (2002). In addition a sensitivity analysis of individual parameters from these state of the art models could be used to help designers further control the affects of axial shortening.

iii) The method is designed so that framing action can be entered by the user. This requires that the user understand the structural mechanics of the building, but it also enables semi-rigid connections to be considered. Refining the actual force-deflection relationship at each connection can be made by further work on plate analysis of concrete slabs with drop panels, and by considering cracking of the beam associated with the moment transfer within the beam. Further work on the effect of transfer truss systems across multiple floors would enable the theory to be applied to a greater range of structures.

iv) The user interface and internal workings of the computer program are basic and cumbersome. While the aim of the research was not to produce a program, more work in this area would make it easier to input and output data.

v) Consideration how prestressing columns or cores will effect framing action requires further research. This has implications in differential axial shortening between column and core, especially with respect to column design.

7.7 Closing Remarks

The problem of axial shortening in columns and cores of TCBs requires careful consideration. A structured approach to the problem is provided for a TCB of general shape and size and of non-specific construction cycle. By developing a model that allows axial shortening calculations for multiple columns and cores, that are linked by any number of framing elements, the author has provided a method for solving an important engineering problem.

REFERENCES

American Concrete Institute 209R-82 (1986) *Prediction of Creep , Shrinkage and temperature effects in Concrete structures.* ACI Manual of Concrete Practice, Michigan.

Bakoss S.L, Brady E.A, Burfitt A.J, Cridland L. (1984) *Long-term Deformations of a 32 Storey High Concrete Building.* RILEM/ACI Volume 11 pages 186-200.

Bazant Z.P (1972) *Prediction of Concrete Effects Using Age-Adjusted Effective Modulus Method.* ACI Journal, Volume 19, Pages 212-217.

Bazant Z.P., Panula L. (1978) *Practical Prediction of Time-Dependent Deformation of Concrete* (Drying creep and Temperature effects) Materials and Structures, Volume 11, Pages 307-328.

Bazant Z.P., Panula L. (1978) *Practical Prediction of Time-Dependent Deformation of Concrete* (Drying creep and Temperature effects) Part 2 Materials and Structures, Volume 11, Pages 415-434.

Bazant Z.P., Panula L. (1979) *Practical Prediction of Time-Dependent Deformation of Concrete* (Temperature effects and cyclic creep) Materials and Structures, Volume 12, Pages 169-183.

Bazant Z.P. (1988) *Mathematical Modeling of Creep and Shrinkage of Concrete.* Anchor Press Ltd, Tiptree, Essex.

Bazant Z.P., Kim J.K. (1991) *Improved Prediction Model for Time-Dependent Deformations of Concrete.* Part 1 Materials and Structures, Volume 24 pages 327-345.

Bazant Z.P., Kim J.K. (1992) *Improved Prediction Model for Time-Dependent Deformations of Concrete.* Part 2 Materials and Structures, Volume 24 pages 409-421.

Bazant Z.P., Kim J.K. (1992) *Improved Prediction Model for Time-Dependent Deformations of Concrete.* Part 3 Materials and Structures, Volume 25 pages 21-28.

Bazant Z.P., Kim J.K. (1992) *Improved Prediction Model for Time-Dependent Deformations of Concrete.* Part 4 Materials and Structures, Volume 25 pages 84-94.

Bazant Z.P., Kim J.K. (1992) *Improved Prediction Model for Time-Dependent Deformations of Concrete.* Part 5 Materials and Structures, Volume 25 pages 163-169.

Bazant Z.P., Kim J.K. (1992) *Improved Prediction Model for Time-Dependent Deformations of Concrete.* Part 6 Materials and Structures, Volume 25 pages 219-223.

Bazant Z.P., Baweja S. (1995) *Creep and Shrinkage Prediction Model for Analysis and Design of Concrete Structures.* Model B3, Materials and Structures, Volume 28 pages 357-365.

Beasley A.J. (1987) *A Numerical Solution for the Prediction of Elastic, Creep, Shrinkage and Thermal Deformations in the Columns and Cores of tall Buildings.* Research Report CM87/2, UTAS.

Beasley A.J. (1987) COLECS: *A Computer Program for the analysis of Elastic, Creep, Shrinkage and Thermal Deformations in the Columns and Cores of tall Buildings.* Research Report CM87/3, UTAS.

Beasley A.J. (1997) *High Performance Concrete- Implications for Axial Shortening in Tall Buildings.* Proceedings of the USA-Australia workshop on High Performance Concrete, Sydney.

British Concrete Society (1978) *A Simplified Method for Estimating the Elastic Modulus and Creep of Normal Weight Concrete,* Cement and Concrete Association Training Centre.

Brooks J.J. (1984) *Accuracy of Estimating Long-term Strains in Concrete.* Magazine of Concrete Research Vol 36 No 128.

Bursle S., Gowripalan N., Baweja D., Kayvani K. (2003) *Refining Differential Shortening Predictions in Tall Buildings for Improved Serviceability.* Conference paper for Concrete Institute of Australia.

Comite Euro-International du Beton (1970) CEB-FIP *International Recommendations for the Design and Construction of Concrete Structures.* CEB, Paris-London.

Comite Euro-International du Beton (1978) CEB-FIP *Code for concrete structures.* CEB, Paris-London.

Davis R.E., Brown E.H. (1937) *Plastic Flow and Volume Changes of Concrete.* American Society for Testing Material Proceeding Volume 37, pages 317-330.

Deitel H.M., Deitel P.J. (1998) *JAVA How to Program.* Prentice Hall, USA.

DIN 4227 (1979) *Teil 1* Spannbeton.

Eckstein R., Loy M., Wood D. (1998) *JAVA Swing.* O Reilly & Associates, USA.

Faber O. (1927) *Plastic Yield, Shrinkage and Other Problems of Concrete and their Effects on Design*, Minutes of Proceedings of the Institute of Civil Engineers, 225, Part 1, London Pages 27-73.

Faber O. (1927) *Plastic Yield, Shrinkage and Other Problems of Concrete and their Effects on Design*, Minutes of Proceedings of the Institute of Civil Engineers, 225, Part 11, London Pages 75-130.

Fintel M., Khan F.R. (1969) *Effects of Column Creep and Shrinkage in Tall Structures-Prediction of Inelastic Column Shortening.* ACI Journal, Volume 66, pages 957-967.

Fintel M., Khan F.R (1971) *Effects of Column Creep and Shrinkage in Tall Structures-Analysis for differential shortening of columns and field observation of structures* Special Publication 27 ACI Journal pages 95-119.

Fintel M., Khan F.R. (1971) *Effects of Column Creep and Shrinkage in Tall Structures-Analysis for Differential Shortening of Columns and Field Observations of Structures. Designing for Effect of Creep, Shrinkage and Temperature in Concrete Structures*, Special Publication Number 27, ACI, Michigan, pages 95-119.

Fintel M., Ghosh S.K. (1984) *High Rise Design. Accounting for Column Length Changes.* Civil Engineering ASCE pages 55-59.

Fintel M. (1986) *Creep and Shrinkage Shortenings in Tall Structures and Their Compensation.* Concrete Institute of Australia, Volume 14 Number 2 pages 11-14.

Gardner M.A. (2003) ASCA Axial Shortening Column Analyser. Program by the author. University of Tasmania.

Gilbert R.I. (1988) *Time Effects in Concrete Structures.* Elsevier science Publishers, New York.

Gilbert R.I. (2002) *Creep and Shrinkage Models for High Strength Concrete.* Journal of Structural Engineering ASCE, Number 2, Volume 4, pages 95-106.

HansenT.C, Mattock A.H. (1966) *Influence of Size and Shape of Members on the Shrinkage and Creep of Concrete.* ACI Journal, Volume 63, Pages 267-290.

Jardin C.A., Dixon P. (1997) *Visual Caf Source Book.* John Wiley & Sons, Canada.

Knudsen J. (1999) *JAVA 2D Graphics.* O Reilly & Associates, USA.

Koutsoukis M., Beasley A.J. (1994) *Monte Carlo Analysis of tall Concrete Structures.* CST Proceedings, Edinburgh.

Koutsoukis M., Beasley A.J (1995) *Axial Shortening Prediction Methods for Tall Concrete Buildings.* Part 1: Comparison of Theoretical Methods. 14 ACMSM proceedings, University of Tasmania, Volume 2 pages 528-534.

Koutsoukis M., Beasley A.J (1995) *Axial Shortening Prediction Methods for Tall Concrete Buildings.* Part 11: Sensitivity Analysis and Experimental Comparisons. 14 ACMSM proceedings, University of Tasmania, Volume 2 pages 535-540.

Koutsoukis M. (1996) *Probabilistic Time-Dependent Axial Shortening of Tall Concrete Buildings* Vol 1. Research Thesis, UTAS.

Koutsoukis M., Beasley A.J (1997) *A Final Report on Axial Shortening Analyses for Tall Concrete Buildings.* University of Tasmania.

McHenry D. (1943) *A New Aspect of Creep in Concrete and its Application to Design.* ASTM Volume 43, Pages 1069-1086.

Martin O., Kayvani K., Marengo L. (2003) *Structural Design of Sydneys Tallest Residential Building.* Proceedings of Advances in Structures (ASSCCA 03) Sydney.

Melerski E.S. (1991) *Simple Elastic Analysis of Axisymmetric cylindrical Storage Tanks.* Journal of Structural Engineering ASCE, Number 11, Volume 117.

Melerski E.S. (1992) *Probabilistic Computer Analysis of Rafts.* Computer and Structures, Number 6, Volume 43.

Melerski E.S. (1995) *Matrix Stiffness Method.* 4th year Structural Mechanics Lecture Material Pages 65-80. University of Tasmania

Mendis P., Hira A. (1999) *Use of High-Strength Concrete in Tall Buildings* International Conference, Innovation in Concrete Structures, UK.

Muller H.S., Hilsdorf H.K. (1982) *Comparison of Prediction Methods for Creep Coefficients of Structural Concrete with Experimental Data.* Martinus Nijhoff, The Hague.

Neville A.M., Dilger W.H., Brooks J.J. (1983) *Creep of Plain and Structural Concrete.* Construction Press, London.

Pauw A. (1960) *Static Modulus of Elasticity of Concrete as Affected by Density.* ACI Journal, Volume 57, Pages 679-687.

Pickett, G. (1946) *The Effect of Change in Moisture Content on the Creep of Concrete Under Sustained Load.* ACI Journal, Volume 42, pages 165-204.

Placzek P. (1997) Comparison of Various methods of utilizing COLECS output in 3D Frame Analysis, MeinHardt Report.

Placzek P. (1997) *Movement Control Report on L10-11 Transfer Truss System*, MeinHardt Report.

Placzek P. (1998) Plots of Predicted and Measured Column and Core Compressions, MeinHardt Report.

Placzek P. (2003) Structural Design Notes and Graphs provided by MeinHardt.

Ross A.D. (1943) *Creep and Shrinkage in Plain, Reinforced and Prestressed Concrete.* Institute of Civil Engineers (London), Volume 21.

Ross A.D. (1958) *Creep of Concrete Under Variable Stress.* ACI Journal Volume 54, Pages 739-758.

Samara R.M. (1989) Creep Model for reinforced Concrete Columns. Journal of Structural Engineering ASCE, Number 3, Volume 121, pages 399-407.

Seed H. B. (1945) *Creep and Shrinkage in Reinforced Concrete Structures.* The Reinforced Concrete Association. Review of Recent Progress, London.

Seed H.B. (1948) *Creep and Shrinkage in Reinforced Concrete Structures.* Reinforced Concrete Association, London. Shilstone J.M. (1994) Paradigm Shifts in Technology for Normal Strength Concrete. ACI Special Publication SP 114-4.

Smerda Z., Kristek V. (1988) *Creep and Shrinkage of Concrete Elements and Structures.* Elsevier Science Publishers, New York.

Standards Australia (1974) *AS 1481 SAA Prestressed Concrete Code.* Standards Association of Australia, Sydney

Standards Australia (1993) *AS 1170 SAA Loading Code.* Standards Association of Australia, Sydney

Standards Australia (2001) *AS 3600 Concrete Structures.* Standards Association of Australia, Sydney

Rusch H., Jungwirth D., Hilsdorf H.K. (1983) *Creep and Shrinkage, Their Effect on the Behaviour of Concrete Structures.* R.R. Donnelley and Sons, Harrisonburg.

Trost H. (1967) *Auswirkungen des superpositionsprinzips auf kriech und relaxations probleme bei beton und spannbeton.* Beton und Stahlbetonbau, Volume 62, pages 230-238.

Troxell G.E., Raphael J.M., Davis R.E. (1958) *Long-Time Creep and Shrinkage Tests of Plain and Reinforced Concrete.* American Society for Testing Material Proceeding Volume 58, pages 1101-1120.

Warner R.F.. (1975) *Axial Shortening in Reinforced Concrete Columns.* Internal University Publication, University of NSW, Kensington.

Warner R.F. (1975) Effect of Beam Interaction on Concrete column Shortening. UNICIV Report 144, University of NSW, Kensington.

Warner R.F. (1976) *Axial Shortening in Reinforced Concrete Columns.* Australian Civil Engineering Transactions. Pages 15-19

Woolside J., Bull D., Sancandi P. (1995) *Moment Resisting Frame Systems.* Reinforced Concrete Digest. No 17. Pages 10-15

SECTION 2

ACKNOWLEDGMENTS

The Acknowledgements here are made to those who contributed towards the production to the Program. All structural design and programming has been written by the author, however technical assistance was provided by the following people to assist with debugging and general programming layout.

Chris Dalton	For his initial assistance towards establishing a programming background in Java.
Glenn Lewis	For much valued advice tutoring me in Java programming, and providing assistance refining the program.
Brian Cousins	For assistance with Excel® programming.

CONTENTS

Section 2

Chapter 1

Chapter 2

Chapter 1

SOFTWARE DEVELOPMENT

1.1 Introduction

Chapter 6 of Section 1 details the methodology and the logic behind the program written to perform axial shortening calculations for framed structures. The computer program was required to perform the shortening and framing calculations that are prohibitive by hand. One of the objectives of the project was to enable axial shortening calculations to be calculated simply, thus the program is to be viewed as the tool that enables calculations to be made readily and with relative ease. It is not however a fully debugged commercial application although it has commercial value. A manual outlining how the program is used and its limitations, as well as the source code is presented in this section as a demonstration of what is required to perform axial shortening calculations. Before beginning to write code, a good understanding of the task and the objectives of the project need to be specified. The objectives are based on the desired application of the program and limitations of the scope of the project. Refer Chapter 6.

1.2 Choice of Language.

The choice of programming language often depends on the existing skills of the programmer, however in this case the programming skills of the author were limited at commencement of the project so a choice was made between a sequential language or a class based language. A sequential programming language relies on procedures to link global variables, such as Fortran or Pascal. This differs drastically from an object orientated program language like C++ or Java. A multi-storey building is made up of a number of components, for example, columns, beams and wall, so an object orientated language lends its self to this type of scenario. Java was chosen for its web based structure and for the existing classes that were available for download. It was also well understood at the University of Tasmania with good tutor support.

1.3 Flow Charts

Figure 1.1 illustrates the processes required to set up the data required to perform axial shortening calculations. Figure 1.2 demonstrates the logic process the program performs to arrive at axial shortening values.

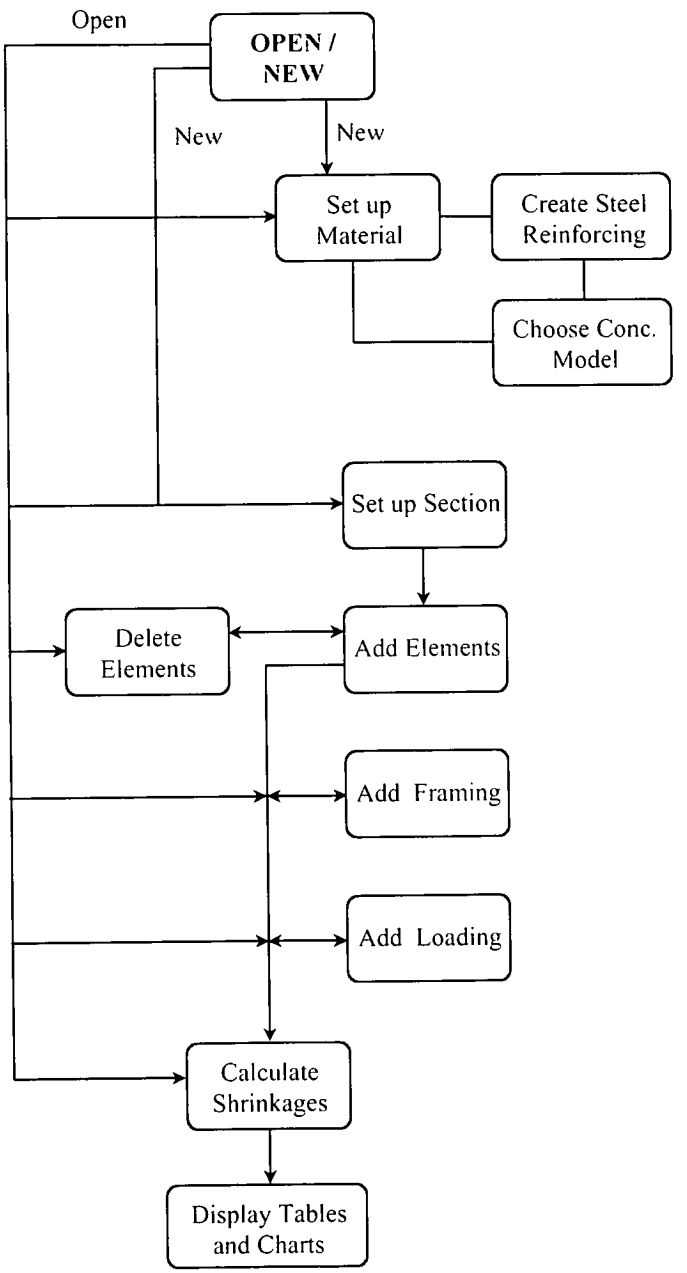


Figure 1.1. Flow chart to set up the building mod

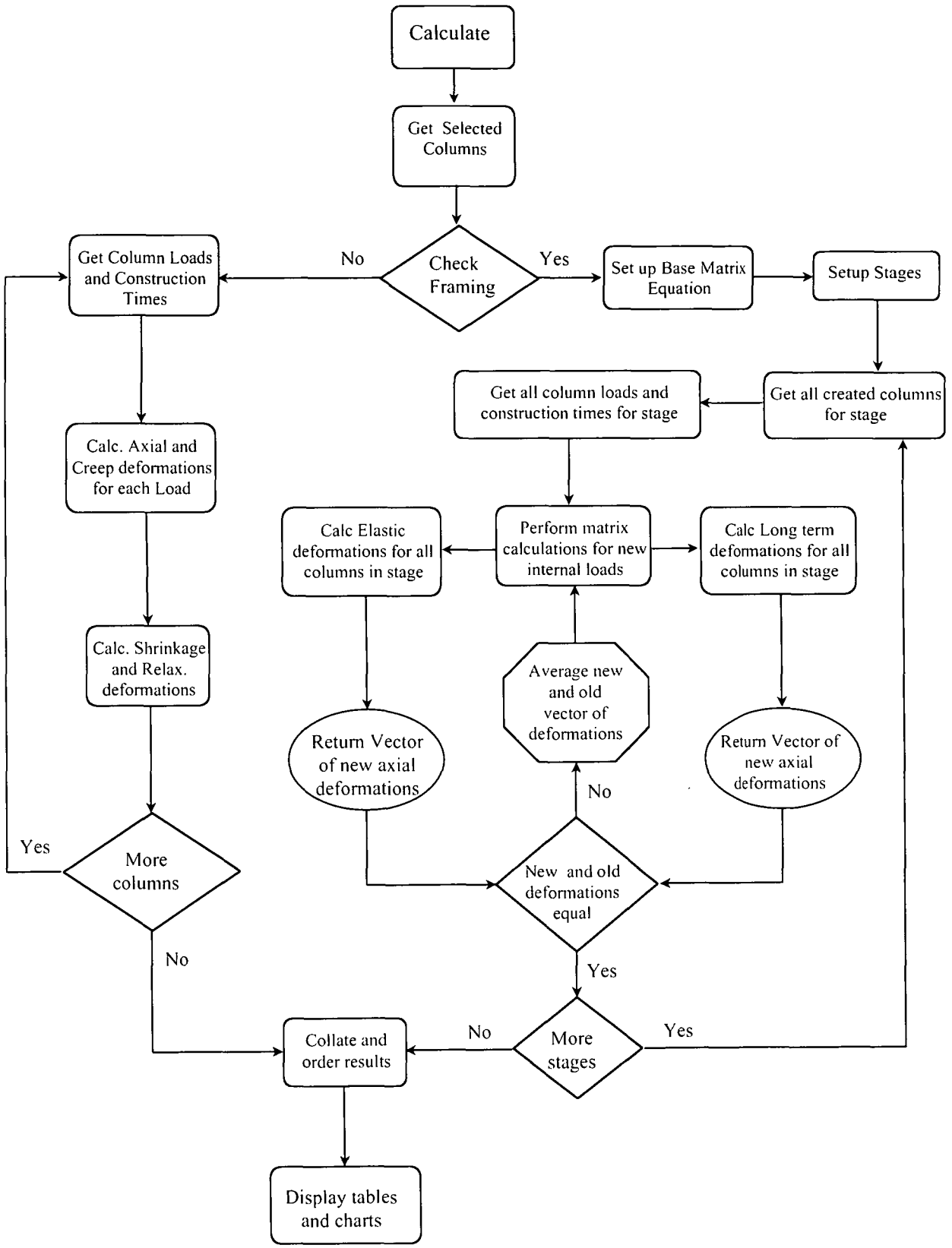


Figure 1.2 Flow chart of logic structure for performing calculations in the program

1.4 Developing the Code

Java is one of the newest object oriented programming languages (Deitel 1998). It is very similar to C++ yet fundamentally different in ways outside the scope of the research. Both languages contain a library of coded classes assisting with the creation of a software package. One of the attractions with Java is that it is one of the preferred languages for use on the World Wide Web (Deitel 1998).

Two aspects of the software needed to be coded. The first aspect was the generation of classes, number crunching and manipulation of classes. The other aspect was to create the graphical user interface for inputting and outputting data (Knudsen 1998). Java contains a number of components that make it possible to create a graphical input interface with outputs in table form (Jardin 1997). The sophistication of the input and output interface is low as the main aim is to produce a working program, not a package for resale. The user interface consists of menus that create dialog boxes. Inputs are prompted and any errors highlighted at time of entry. Entering information can be performed in any order[1], however, the most logical way is to follow the steps at each menu from left to right, top to bottom. The next sections depicts a graphic account of the steps required to input data, perform calculations and output data, as well as some of the associated error controls.

1.4.1 Data inputting procedure

To begin, a new structure can be created, or an old structure opened. This is performed via the file menu. The file menu works in a similar way to any other programs file menu. Refer figure 1.1.

[1] The program will only allow some inputs after other related information is entered

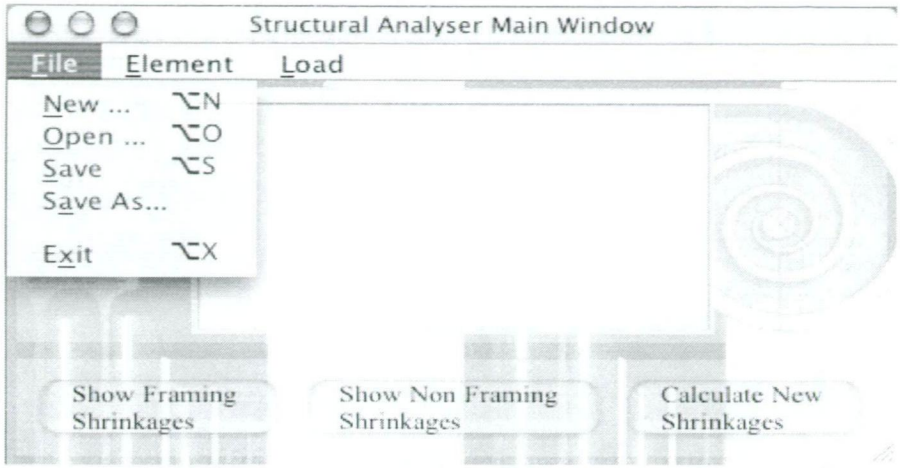


Figure 1.1 Starting Menu

Before a structure can be created, material properties and section properties need to be entered. Any number of each can be defined. Both are found under the Element Menu. Notice how the other functions of the menu are disallowed.

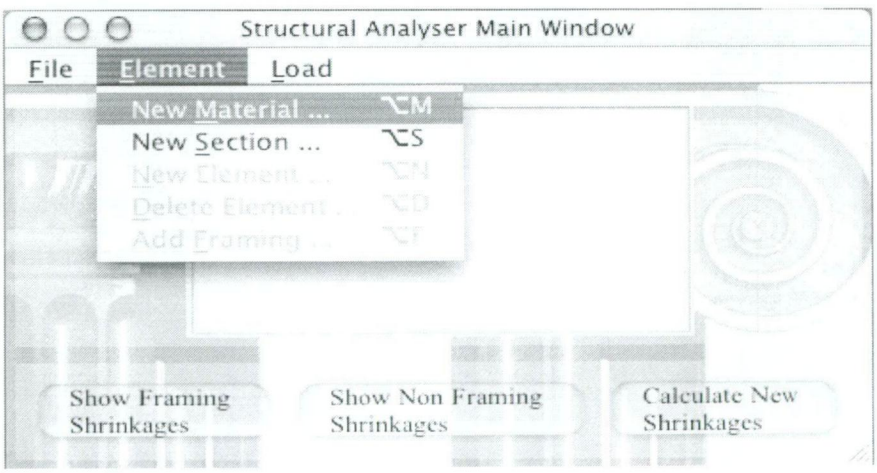


Figure 1.2 Elements menu where the building structure is defined.

The Element menu calls the Materials Editor (Figure 1.3) and Section Editor (Figure 1.4). All the concrete properties are entered as required for the concrete model that is chosen to calculate axial shortening. The Program is written for the ACI(1986) model, however other models can easily be added to the program.

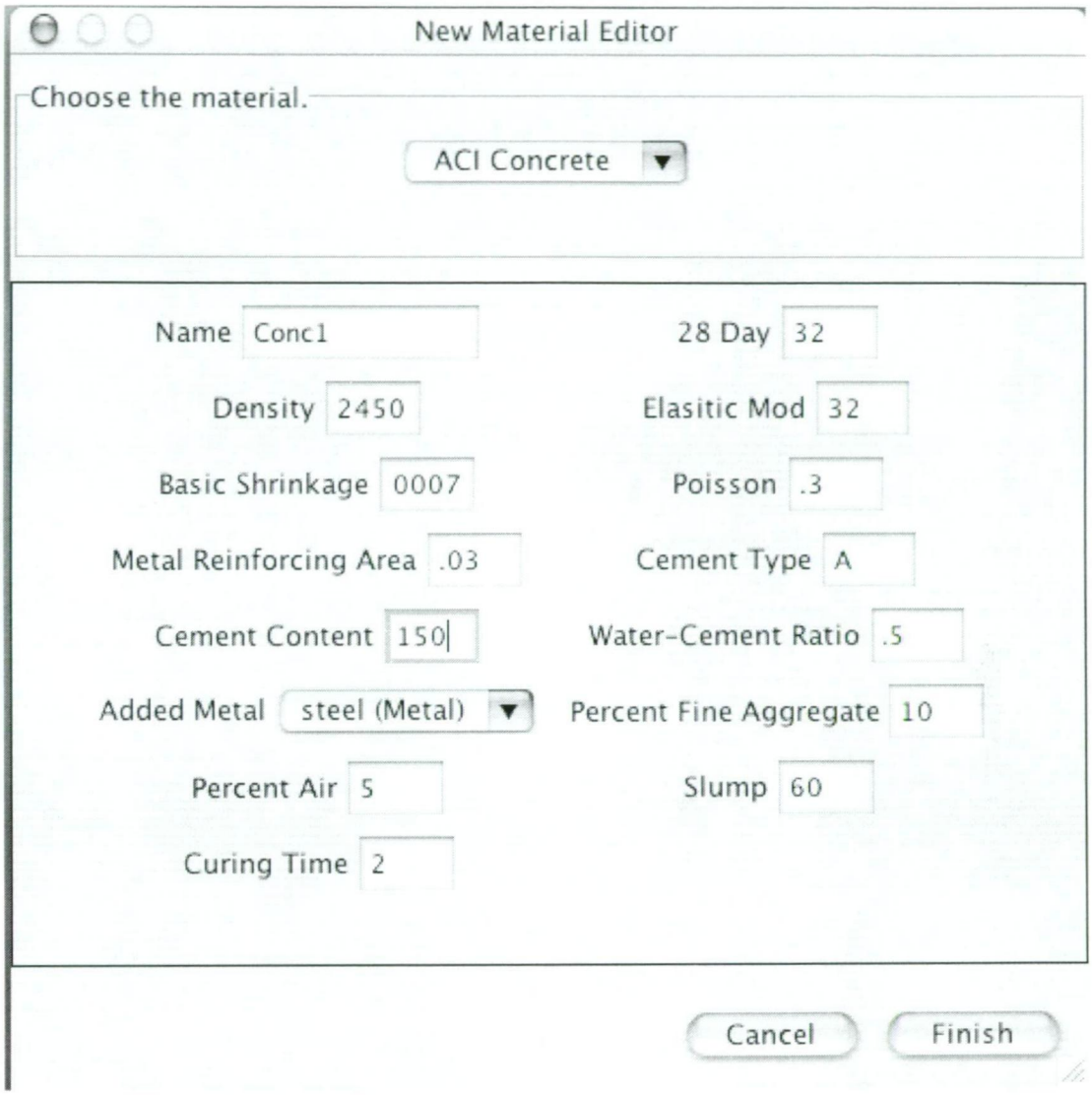


Figure 1.3 Materials Input Dialog box showing parameter inputs for ACI concrete model

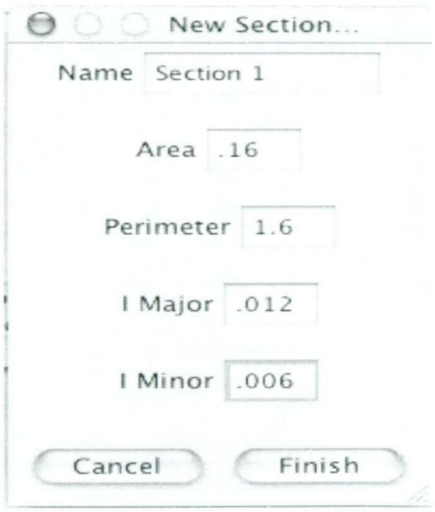


Figure 1.4 Section Input Dialog Box

With a material and a section entered, an element can be created using the material and section.

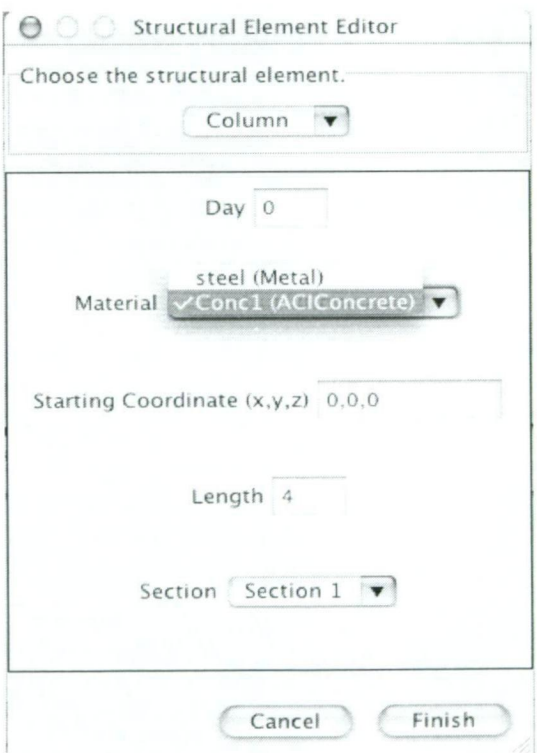


Figure 1.5 Element Editor creating a new column

The Element Editor will automatically calculate a new element with the next available sequential ID based on the inputs provided. In Figure 1.6, a column constructed on day (0), and coordinate (0,0,0)[2] with material of type *Conc1* and *Section 1* will be created.

Once a number of columns have been created, the framing relationship between individual columns is entered. This is entered at each floor level on the desired day and is the stiffness relationship between two columns at each connection. Figure 1.6 and 1.7 show four columns created over two levels. The Framing Editor in Figure 1.7 allows for different stiffness relationships between columns to be calculated by the user and entered at each connection.

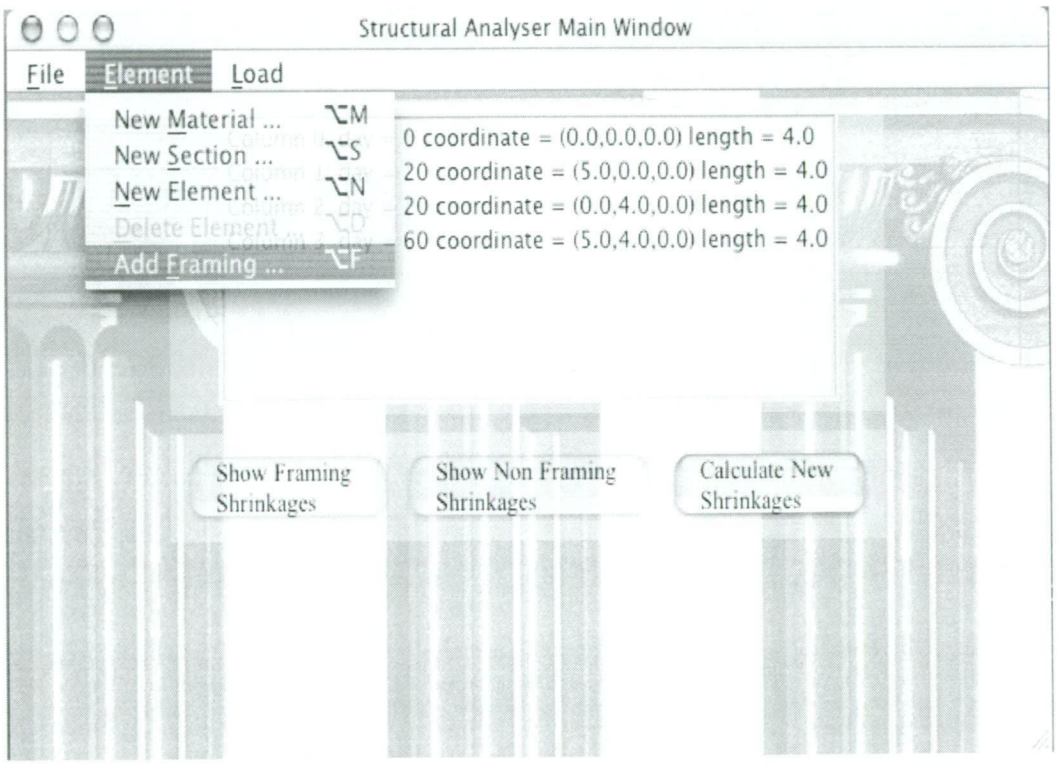


Figure 1.6 Main Window showing 4 columns and the Add Framing menu

[2] Coordinated are global x,y,z coordinates. x and z are in plan, and y is the height

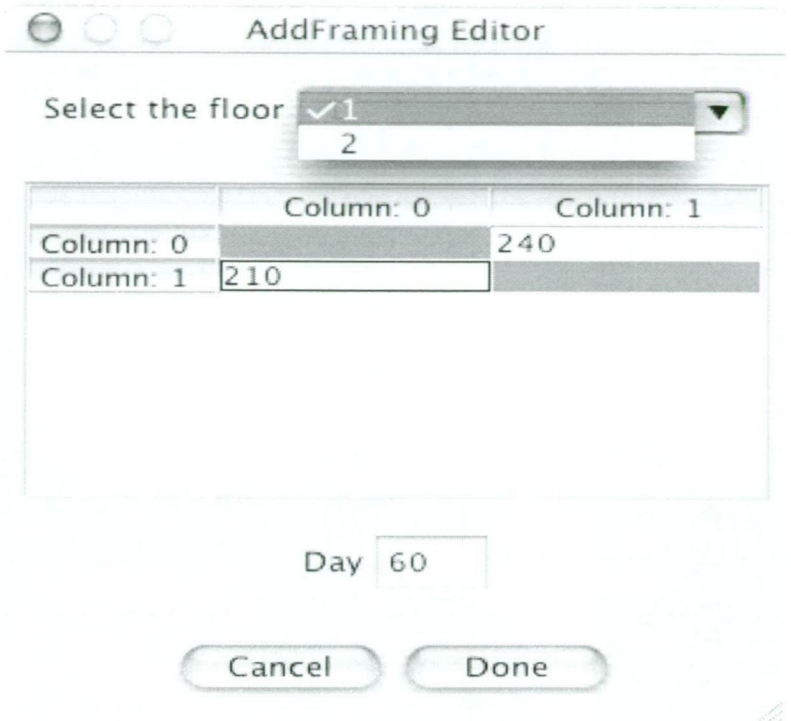


Figure 1.7 Framing Editor showing the link between Column 0 and Column 1 on the first floor

In the structure set up here, there are only 2 columns on each of the first and second floors. Figure 1.7 shows different stiffness relationship values at each column. At the connection to column 0 from the connection element between Column 0 and Column 1, the stiffness value is 210 kN/m [3], while at the connection to Column 1 from the same member, the stiffness value is 240 kN/m.

Finally the applied loads need to be entered. Individual dead loads for each column are automatically calculated and recorded at the time created. Additional loads can be entered at any time after the creation day of the column in question. This can be done via the menu or by double clicking on a particular column. Figure 1.8 below shows a typical load being created.

[3] The stiffness values entered in Figure 6.9 are purely arbitrary

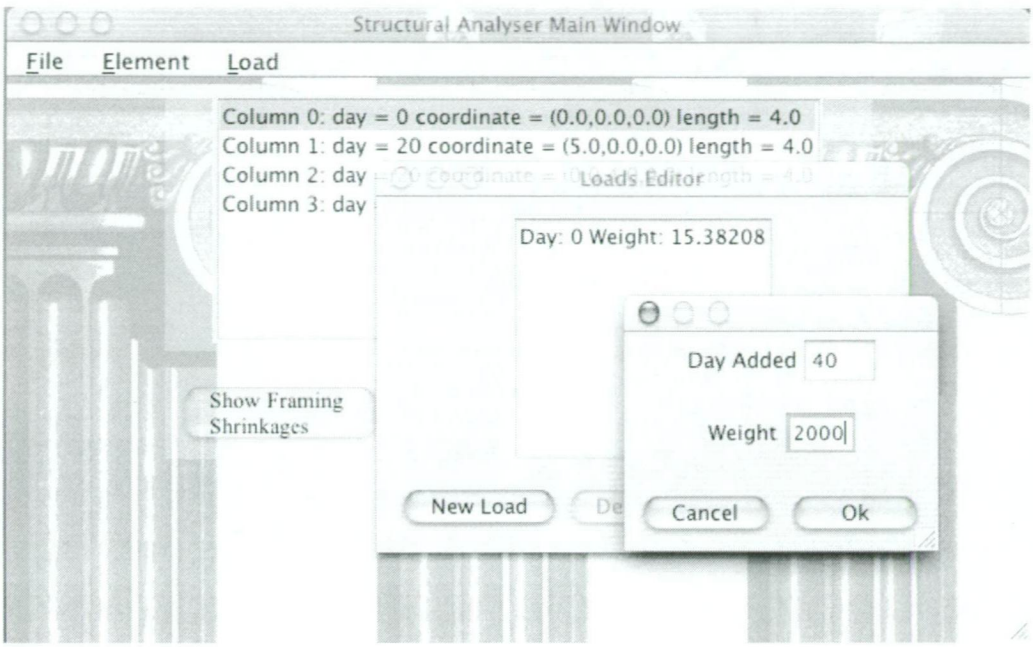


Figure 1.8 Load Editor with a 2000kN load being added to Column 0 at day 40

The information entered is sufficient for calculations of axial deformations for both framed and isolated columns to be made at any desired time. The point of time that calculations are made does not have to be after the entire structure has been constructed. Elements that have not yet been created will not have a deformation value (note the deformation values are global and are calculated as a total deformation at the top of the column relative to the ground).

1.4.2 Errors

When writing a program as large and as complex as the one used in this research, mistakes will be made that impact on the final results. Given that finding and debugging a program so large can be laborious, at all times the validity and accuracy of interim and test results needs to be questioned.

Errors in the program can be considered in two areas. Firstly in respect to the underlying theory and how that is constructed within the program and, secondly, whether there are gross errors in the programs implementation (Eckstein 1998). The latter set of errors will generally produce equally gross errors in output.

i) errors in the application of the developed theory can be placed into two categories;
   a) program accuracy when dealing with numeric equations and variable limits
   b) intrinsic application of the theory to code and the general flow of the program

ii) gross errors are generated by incorrect coding, uncontrolled loops or the generation of undefined variables or results

Four approaches are adopted to validate the program results ;

i) catching errors at input
ii) catching generated errors
iii) breaking the program into workable parts that can each be tested separately
iv) testing and calibrating the program at output stage

Catching errors at input or when generated by the program are functions of the program language and is essentially good programming practice.

There are three methods the program utilises to stop inputs that will cause errors in calculations. The first is discussed above where the user is limited as to what functions they can use, so that variables that rely on data from other major classes cannot be created until all data is available.

Secondly is a *try and catch* structure within the program that checks calculations for errors (Deitel 1998). If detected the program will either terminate the calculation procedure and return an error dialog or adopt a default value.

Thirdly is the catch method where the program bans the inputting of information that is logically impossible. An example of the program displays such errors at input stage is shown below.

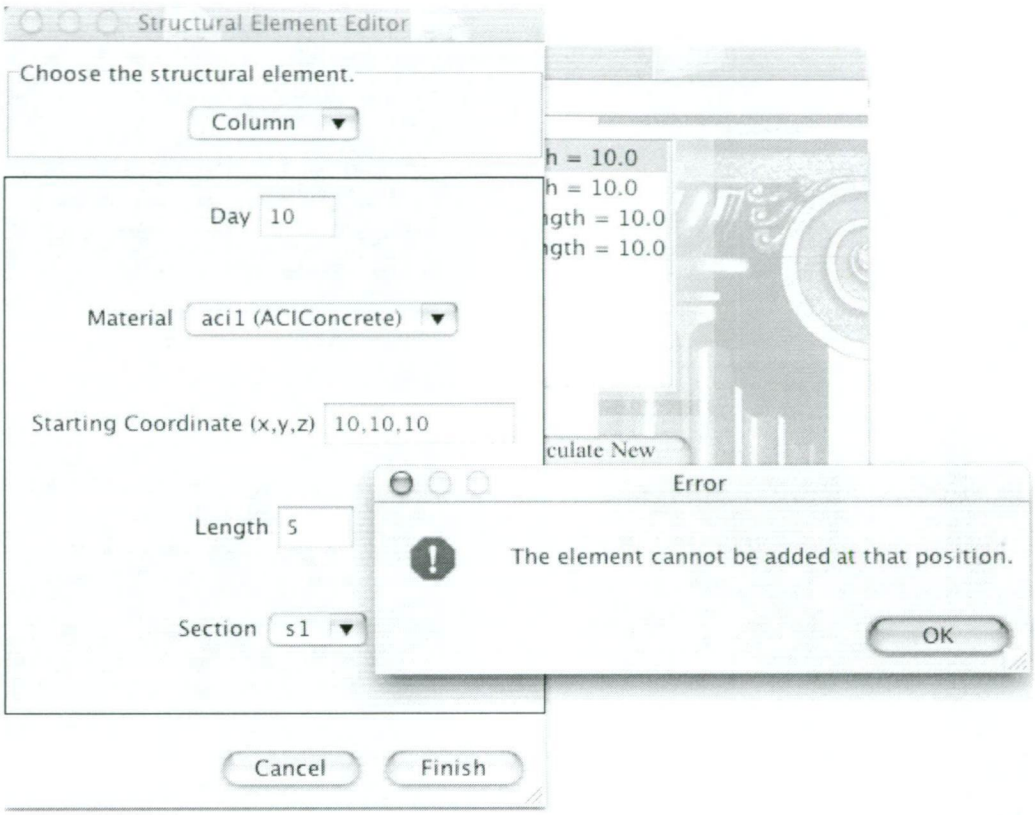


Figure 1.9 Attempting to add an element at an incorrect position

Breaking the program into small workable parts is also good programming practice, and can be made easier by the programming language (Deitel 1998). Object oriented programs such as Java are almost a collection of mini programs that are all linked, so compiling and testing of smaller parts is made easy with the adopted programming language.

Calibrating the results of a program is essential to fully validate results. If the program produces unique results then calibration may not be possible. In the case of this research, a number of small models are coded as sub routines that can each be calibrated with actual

results (Deitel 1998). The results from a framing analysis are an extension of results from existing programs like COLECS so the program is written to allow the developed theory to be manipulated to shadow the existing program and thus produce results that can be calibrated with previously tested material. The final set of results can then be presented with far more confidence. Chapter 5 deals with how this is done in more detail.

1.4.3 Calculating and outputs

Calculations can be made at any time and do not rely on the full structure being completed. However the user needs to be wary of making calculations and then adding elements at times earlier that previously calculated. As a safe guard, if a calculation is repeated for the same day as a previous calculation, the program will override the new data for the latest calculation.

To perform a calculation, select the *Calculate New Shrinkages* button to generate the Calc Delta Dialog as shown in Figure 6.12 below. Different elements on different planes or levels can be selected by choosing a specific coordinate value or, alternatively all elements can be selected by choosing all coordinates. Here the user can select to analyse with or without framing. The reason for not choosing framing is that in some structures there may not be significant framing action and results without framing are generated in a much shorter amount of time. This also enables the program to be calibrated against another program that produces tested results without framing effects, such as COLECS (1987).

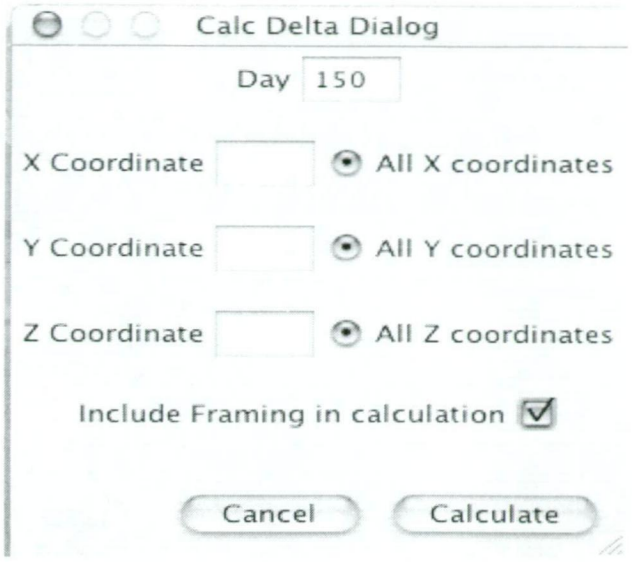


Figure 1.10 Calc delta Dialog box

The program produces a table and saves the data in an Excel® file where graphs can be created. Figure 1.11 is an example results table generated by ASCA(2003) and Figure 1.12 is an example of a chart generated from Excel® (both are examples only).

Results Table

Column	Day 150	Day 200	Day 300
Column 0	13.64003...	14.29464...	14.97851...
Column 1	6.632308...	7.080320...	7.571575...
Column 3		18.93278...	19.88006...
Column 4		12.67984...	13.58316...

Figure 1.11 Table of results for 4 columns

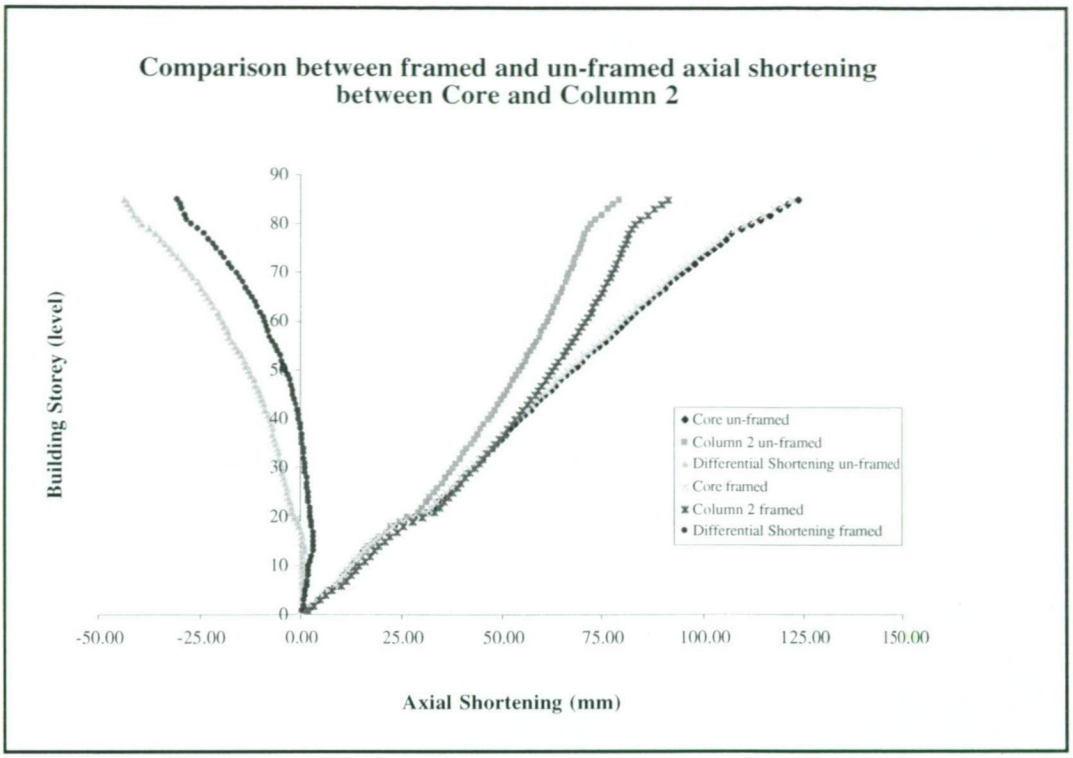


Figure 1.12Excel® Chart for two columns of the Bayoke Tower II

1.5 Source Code.

As object orientated programs are made of many smaller classes which in themselves are like mini programs. The source code presented is given in alphabetic order of class names. This does not reflect the order of operation of the program. The first class called when the package is opened is the StructuralAnalyser.java class.

Chapter 2

CODE

```
/*
 * @(#)AIConcrete.java
 *
 * History:  1999-2000: Written by M. Gardner and C.Dalton.
 *           Oct 2001:  G.Lewis added comments, deleted obsolete code,
 *                and changed variable names.
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser.material;

import java.io.*;
import structuralAnalyser.*;

/** The class represents the material ACI Concrete */
public class ACIConcrete extends Concrete implements Serializable
{
  /** The percentage fine aggregate */
  protected float myPercentFineAggregate;
  /** The percentage air */
  protected float myPercentAir;
  /** The slump */
  protected float mySlump;
  /** The curing time */
  protected float myCuringTime;

  /**Constructor
   * @param id the unique name of this material */
  public ACIConcrete(String id, float conc28day, float density,
                float elasticMod, float thermalExp,
                float humidity, float metalReinforcingArea,
                String cementType, float cementContent,
                float waterCementRatio, Metal addedMetal,
                float percentFineAggregate, float percentAir, float slump,
                float curingTime)
    throws InvalidParameterException
  {
    super(id, "ACIConcrete", conc28day,density, elasticMod, thermalExp,
humidity,
        metalReinforcingArea, cementType, cementContent, waterCementRatio,
        addedMetal);
    setPercentFineAggregate(percentFineAggregate);
    setPercentAir(percentAir);
    setSlump(slump);
    setCuringTime(curingTime);
  }
```

```
  /** Get the percentage fine aggregate. Returns a float between 0 and 100. */
  public float getPercentFineAggregate()
  {
    return myPercentFineAggregate;
  }
  /** Set the percentage fine aggregate.*/
  public void setPercentFineAggregate(float percentFineAggregate)
    throws InvalidParameterException
  {
    if ((percentFineAggregate < 0) || (percentFineAggregate  > 100))
      throw new InvalidParameterException("percent fine aggregate must be
between 0 and 100.");
    myPercentFineAggregate = percentFineAggregate;
  }
  /** Get the percentage air. Returns a float between 0 and 100.*/
  public float getPercentAir()
  {
    return myPercentAir;
  }
  /**Set the percentage air. */
  public void setPercentAir(float percentAir) throws InvalidParameterException
  {
    if ((percentAir < 0)||(percentAir > 100))
      throw new InvalidParameterException("percent air aggregate must be
between 0 and 100.");

    myPercentAir = percentAir;
  }
  /**Get the slump */
  public float getSlump()
  {
    return mySlump;
  }

  /**Set the slump */
  public void setSlump(float slump)
  {
    mySlump = slump;
  }

  /**Get the curing time */
  public float getCuringTime()
  {
    return myCuringTime;
  }

  /**Set the curing time */
  public void setCuringTime(float curingTime) throws InvalidParameterException
  {
    if (curingTime <= 0)
      throw new InvalidParameterException("The curing time must be greater
than 0.");
    myCuringTime = curingTime;
  }
}
```

```
/*
 * @(#)AddFramingDialog.java
 *
 * History:  Mar 2002:  Written by M. Gardner and G. Lewis
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.awt.BorderLayout;
import java.awt.Dimension;
import javax.swing.table.*;
import Jama.*; //matrix package, from http://math.nist.gov/javanumerics/jama/

import structuralAnalyser.structuralElement.*;
import structuralAnalyser.material.*;

/** The dialog for entering framing details. */
public class AddFramingDialog extends JDialog {
    /** The pane that displays the floor selection */
    FloorSelectionPane myFloorSelectionPane = null;
    /**The pane that displays the buttons (cancel, done) */
    ButtonsPane myButtonsPane = null;
    /**The panel that display the tables for entering framing details*/
    JPanel tablePane = null;
    /**A vector of all the tables */
    Vector tablePanelVector = null;
    //Actions used in all panes
    /** Cancel action */
    Action myCancelAction  = new CancelAction();
    /** Done action */
    Action myDoneAction  = new DoneAction();
    //Borders used in all panes.
    /**A border that puts 10 extra pixels around a pane.*/
    Border myPaneEdgeBdr = BorderFactory.createEmptyBorder(10,10,10,10);
    /**Etched border, used for adding border to components.*/
    Border myEtchedBdr = BorderFactory.createEtchedBorder();


    /**Constructor
     *  @param parent the owner of this dialog
     */
    public AddFramingDialog(JFrame parent) {
      super(parent, "AddFraming Editor", true);

      JPanel content = new JPanel();
      content.setLayout(new BoxLayout(content, BoxLayout.Y_AXIS));
      content.setOpaque(false);
```

```
        myFloorSelectionPane = new FloorSelectionPane();
        content.add(myFloorSelectionPane);
        tablePane = new JPanel();
          tablePane.setLayout(new CardLayout());
        tablePanelVector = new Vector();

        int numFloors = Structure.getInstance().getNumFloors();
        //add framing table for each floor except the ground floor
        for (int i =1; i < numFloors; i++)
            tablePane.add(constructTable(i,tablePanelVector),
                      Integer.toString(i));

        content.add(Box.createRigidArea(new Dimension(0, 10)));
          content.add(tablePane);

        myButtonsPane = new ButtonsPane();
        content.add(Box.createRigidArea(new Dimension(0, 10)));
        content.add(myButtonsPane);

        getContentPane().setLayout(new FlowLayout());
        getContentPane().add(content);
      }

      /** Construct the table for a given floor. Retrieve values from
       * the structure if possible.
       * @param floorNumber the floor to construct the table for
       * @param numFloors the number of floors of the structure
       * @param tablePanelVector store table panel for each table
       */
      public JPanel constructTable(int floorNumber,
                           Vector tablePanelVector){

        //the floor number should be at least the first floor. We use
        //the columns supporting the floor for framing. For example,
        //framing for floor 1 uses the columns on the ground floor
        Assert.assert(floorNumber > 0);

        System.out.println(" here in construct table");
System.out.println(floorNumber + " floorNumber");
//System.out.println(tablePanelVector + " tablePanelVector");
        Vector columns =
            Structure.getInstance().getColumns(new Integer(floorNumber-1));
        System.out.println(columns + " columns");
        DefaultTableModel headerData = new DefaultTableModel(0, 0);
        headerData.addColumn("");
        Vector columnNames = new Vector();
        //set up the column and row headers
        for (int i = 0; i < columns.size(); i++){
          System.out.println(" here in loop");
            Column col = (Column) columns.elementAt(i);
            headerData.addRow(new Object[] {"Column: "+ col.getId()} );
            columnNames.addElement("Column: "+ col.getId());
        }
        JTable rowHeader = new JTable(headerData);

        Framing framing =
            Structure.getInstance().getFraming(new Integer(floorNumber));
```

```
  DefaultTableModel tableModel =
      new DefaultTableModel(columnNames,columns.size()){
        public boolean isCellEditable(int row, int column){
            return row != column;
        }
      };
  //set the framing data in the table
  for (int n =0; n < framing.getNumberColumns(); n++){
      for (int m =0; m < framing.getNumberRows(); m++){
        double dblValue = framing.getValue(m,n);
        if (dblValue != 0){
            String strValue = Double.toString(dblValue);
            Assert.assert(strValue != null);
            tableModel.setValueAt(strValue,m,n);
        }
      }
  }
  JTable table = new JTable(tableModel);
  table.setDefaultRenderer(Object.class, new FramingTableCellRenderer());

  LookAndFeel.installColorsAndFont(rowHeader, "TableHeader.background",
                           "TableHeader.foreground",
                           "TableHeader.font");

  rowHeader.setIntercellSpacing(new Dimension(0, 0));
  Dimension d = rowHeader.getPreferredScrollableViewportSize();
  d.width = rowHeader.getPreferredSize().width;
  rowHeader.setPreferredScrollableViewportSize(d);
  rowHeader.setRowHeight(table.getRowHeight());
  rowHeader.setDefaultRenderer(Object.class, new RowHeaderRenderer());

  TablePanel tablePanel = new TablePanel(table,rowHeader,framing);
  tablePanelVector.addElement(tablePanel);
  return tablePanel;
}
/**Inner class for table and day panel */
public class TablePanel extends JPanel{
  /**The table displayed in this table panel */
  JTable myTable = null;
  /**The day the framing was applied*/
  int myDayFramingAdded = -1;
  /**The day the element was constructed */
  private JLabel myDayLabel = null;
  private JTextField myDayField = null;

  /**Constructor
   * @param table the table for entering framing
   * @param rowHeader the row headers for the table
   * @param framing the framing for the table
   */
  public TablePanel(JTable table, JTable rowHeader, Framing framing){
      JScrollPane scrollPane = new JScrollPane(table);
      scrollPane.setRowHeaderView(rowHeader);
      JTableHeader corner = rowHeader.getTableHeader();
      corner.setReorderingAllowed(false);
      corner.setResizingAllowed(false);
      scrollPane.setCorner(JScrollPane.UPPER_LEFT_CORNER, corner);
```

```
        setPreferredSize(new Dimension (300,200));
        myTable = table;
        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
        add(scrollPane);
        JPanel dayPanel = new JPanel();
        myDayLabel = new JLabel("Day");
        myDayField = new JTextField(4);
        if (framing.getDayApplied() > 0)
          myDayField.setText(Integer.toString(framing.getDayApplied()));
        dayPanel.add(myDayLabel);
        dayPanel.add(myDayField);
        add(Box.createRigidArea(new Dimension(0, 10)));
        add(dayPanel);
    }
    /**Get the table of this table panel */
    public JTable getTable(){
        return myTable;
    }
    /**Get the day of this panel, or -1 if the day is invalid */
    public int getDayApplied(){
        String dayString = myDayField.getText();
        int day = -1;
        try{
          day = Integer.parseInt(dayString);
        }
        catch (NumberFormatException e){
          day = -1;
        }
        return day;
    }
}
/**Inner classs for framing table cell renderer */
public class FramingTableCellRenderer extends DefaultTableCellRenderer {

    public FramingTableCellRenderer() {}

    public Component getTableCellRendererComponent
        (JTable table, Object value, boolean isSelected,
         boolean hasFocus, int row, int column)
    {
        Component cell = super.getTableCellRendererComponent
          (table, value, isSelected, hasFocus, row, column);
        if (row == column)
          setBackground(Color.gray);
        else
          setBackground(Color.white);
        return cell;
    }
}

/** Inner class for the floor selection panel
 */
private class FloorSelectionPane extends JPanel{
    /** Floor combo box label */
    JLabel myFloorLabel;
    /** Floor number drop down list */
    JComboBox myFloorComboBox;
```

```
  /** The action performed when the list is changed */
  Action myFloorChangeAction;

  /**Constructor */
  public FloorSelectionPane(){
      myFloorLabel = new JLabel("Select the floor");
      int numFloors = Structure.getInstance().getNumFloors();
      Vector allFloors = new Vector();
      for (int i = 1; i < numFloors; i++)
        allFloors.addElement(new Integer(i));
      JComboBox myFloorComboBox = new JComboBox(allFloors);
      myFloorChangeAction = new FloorChangeAction();
      myFloorComboBox.addActionListener(myFloorChangeAction);
      setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
      setBorder(myPaneEdgeBdr);
      add(myFloorLabel);
      add(Box.createRigidArea(new Dimension(5, 0)));
      add(myFloorComboBox);
  }

  private class FloorChangeAction extends AbstractAction{
      public void actionPerformed(ActionEvent e) {
            JComboBox cb = (JComboBox)e.getSource();
            Integer floor = (Integer)cb.getSelectedItem();
        CardLayout cl = (CardLayout)(tablePane.getLayout());
        cl.show(tablePane, floor.toString());
        }
  }
}
/** Inner class for the buttons pane. This pane displays the Add,
 *  Cancel, and Done buttons.
 */
private class ButtonsPane extends JPanel{
  /** Cancel Button */
  JButton myCancelBtn = null;
  /** Done Button */
  JButton myDoneBtn = null;

  /**Constructor */
  public ButtonsPane(){
      myCancelBtn = new JButton("Cancel");
      myCancelBtn.addActionListener(myCancelAction);
      myCancelBtn.setToolTipText("Click to cancel.");

      myDoneBtn = new JButton("Done");
      myDoneBtn.addActionListener(myDoneAction);
      myDoneBtn.setToolTipText("Click when finished.");

      // add the buttons
      setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
      setBorder(myPaneEdgeBdr);
      add(myCancelBtn);
      add(Box.createRigidArea(new Dimension(5, 0)));
      add(myDoneBtn);
  }
}
/**
```

```
     * Inner class that defines the cancel action
     */
    private class CancelAction extends AbstractAction {
      public void actionPerformed(ActionEvent evt) {
          setVisible(false);
      }
    }
    /**
     * Inner class that defines the done action
     */
    private class DoneAction extends AbstractAction {
      public void actionPerformed(ActionEvent evt) {
          //copy the framing data into the structure the index of
          //the tablemodel in the vector is the floor for the
          //framing
          int numFloors = Structure.getInstance().getNumFloors();
          for (int i=0; i < tablePanelVector.size(); i++){
            TablePanel tablePanel =
                (TablePanel) tablePanelVector.elementAt(i);
            JTable table = tablePanel.getTable();
            DefaultCellEditor ce = (DefaultCellEditor) table.getCellEditor();
            if (ce != null)
                ce.stopCellEditing();
            DefaultTableModel tableModel =
                (DefaultTableModel) table.getModel();
            int dayApplied = tablePanel.getDayApplied();

            if (dayApplied != -1){
                Framing framing = null;
                try{
                  framing = new Framing(i+1,tableModel.getColumnCount());
                  framing.setDayApplied(dayApplied);
                  for (int n=0; n < tableModel.getColumnCount(); n++){
                      for (int m=0; m < tableModel.getRowCount(); m++){
                        String strValue = (String) tableModel.getValueAt(m,n);
                        if (strValue != null){
                            double value = Double.parseDouble(strValue);
                            framing.setValue(m,n,value);
                        }
                      }
                  }
                }
                catch (NumberFormatException e){
                  System.err.println("Invalid framing data");
                  System.exit(-1);
                }
                catch (InvalidParameterException e){
                  System.err.println(e.getMessage());
                  System.exit(-1);
                }
                Structure.getInstance().setFraming(new Integer(i+1),framing);
            }
          }
          setVisible(false);
      }
    }
}
```

```
/*
 * @(#)AnalyserFrame.java
 *
 * History:  Nov 2001: Written by M. Gardner and G. Lewis
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

import java.util.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;

import structuralAnalyser.structuralElement.*;
import structuralAnalyser.material.*;

/** The main frame for the structural analyser */
public class AnalyserFrame extends JFrame {
  /**The path to the file for saving*/
  String myFilePath;

  /**The menu bar */
  AnalyserMenuBar myAnalyserMenuBar = null;
  /**The pane that displays the buttons (new element, delete) */
  ButtonsPane myButtonsPane = null;
  /**The pane that displays the list of elements in the structure */
  ElementsPane myElementsPane = null;

  //Actions used in all panes
  /** New element action */
  Action myNewElementAction  = new NewElementAction();
  /** edit element action */
  Action myEditElementAction  = new EditElementAction();
  /** Delete action */
  Action myDeleteAction  = new DeleteAction();
  /**Edit loads action */
  Action myEditLoadsAction = new EditLoadsAction();
  /**New Material action */
  Action myNewMaterialAction = new NewMaterialAction();
  /**New Section action */
  Action myNewSectionAction = new NewSectionAction();
  /** Show framing shrinkage action */
  Action myShowFramingShrinkageAction  = new ShowFramingShrinkageAction();
/** Show shrinkage action */
  Action myShowNonFramingShrinkageAction  =
      new ShowNonFramingShrinkageAction();
  /** Calc shrinkage action */
  Action myCalcShrinkageAction  = new CalcShrinkageAction();
  /** Add Framing action */
  Action myAddFramingAction = new AddFramingAction();
  /** New action */
```

```
Action myNewAction = new NewAction();
/** Open action */
Action myOpenAction = new OpenAction();
/** Save action */
Action mySaveAction = new SaveAction();
/** Save As action */
Action mySaveAsAction = new SaveAsAction();
/** Exit action */
Action myExitAction = new ExitAction();

//Borders used in all panes.
/**A border that puts 10 extra pixels around a pane.*/
Border myPaneEdgeBdr = BorderFactory.createEmptyBorder(10,10,10,10);
/**Etched border, used for adding border to components.*/
Border myEtchedBdr = BorderFactory.createEtchedBorder();

/** Update the state of the buttons */
Runnable refresh = new Runnable() {
    public void run(){
      myButtonsPane.refresh();
      myAnalyserMenuBar.refresh();
    }
  };

/**Constructor */
public AnalyserFrame() {

       super("Structural Analyser Main Window");

  myAnalyserMenuBar = new AnalyserMenuBar();
  setJMenuBar(myAnalyserMenuBar);

  JPanel content = new JPanel();
  content.setLayout(new BoxLayout(content, BoxLayout.Y_AXIS));
  content.setOpaque(false);

  myElementsPane = new ElementsPane();
  myElementsPane.setBackground(new Color(255,255,255,80));
  content.add(myElementsPane);
  content.add(Box.createRigidArea(new Dimension(0, 10)));

  myButtonsPane = new ButtonsPane();
  myButtonsPane.setBackground(new Color(255,255,255,80));
  content.add(myButtonsPane);

  getContentPane().setLayout(new FlowLayout());
  getContentPane().add(content);
  ((JPanel)getContentPane()).setOpaque(false);

  final ImageIcon image = new ImageIcon("images/column.gif");
  final int winc = image.getIconWidth();
  final int hinc = image.getIconHeight();
  JLabel backlabel = new JLabel("");
  if (image.getIconWidth() > 0 && image.getIconHeight() > 0){
      backlabel = new JLabel() {
          public void paintComponent(Graphics g) {
            int w = getParent().getWidth();
```

```
            int h = getParent().getHeight();
            for (int i=0;i<h+hinc;i=i+hinc)
              {
                for (int j=0;j<w+winc;j=j+winc)
                  {
                    image.paintIcon(this,g,j,i);
                  }
              }
          }
        };
    }

  getLayeredPane().add(backlabel, new Integer(Integer.MIN_VALUE));
  backlabel.setBounds(0,0,5000,5000);
  pack();
  setVisible(true);
  refresh();
}
  /**refresh the buttons and menu items
   */
  public void refresh(){
      SwingUtilities.invokeLater(refresh);
  }

/**Add an element to the list of elements being display */
public void addElement(StructuralElement elt){
  myElementsPane.addElement(elt);
}

/**Clear all elements in the list being displayed*/
public void clear(){
  myElementsPane.clear();
}

  /**display all calcuated shrinkages
   * @param parent the parent of the shrinkages table dialog
   * @param framing if true then show the framing shrinkages
   */
  public static void displayShrinkages(Window parent, boolean framing){
      //display the table
      TableDialog dialog;

      if (parent instanceof JDialog)
          dialog = new TableDialog((JDialog) parent);
      else
          dialog = new TableDialog((JFrame) parent);

      //create the first column (headings only)
      Vector headings = new Vector();
      Vector allElements = Structure.getInstance().getAllElements();
      for (int i =0; i < allElements.size(); i++){
          if (allElements.elementAt(i) instanceof Column){
              Column c = (Column) allElements.elementAt(i);
              headings.addElement(c);
          }
      }
      dialog.addColumn("Column",headings);
```

```
        //for each shrinkage day add the shrinkages
        SortedSet shrinkageDays =
            Structure.getInstance().getShrinkageDays(framing);
        Iterator iter = shrinkageDays.iterator();
        while (iter.hasNext()){
            Vector tableColumnData = new Vector();
            int shrinkageDay = ((Integer) iter.next()).intValue();
            Vector shrinkages =
Structure.getInstance().getShrinkages(shrinkageDay,framing);
            tableColumnData.addAll(shrinkages);
            dialog.addColumn("Day " + shrinkageDay, tableColumnData);
        }


        //sort the shrinkages into stacks of columns
        Vector groundFloorColumns =
            Structure.getInstance().getColumns(new Integer(0));
        int rowPos = 0; //the number of sorted rows
        int numRows = dialog.getRowCount();
        int lastDay =
            ((Integer) Structure.getInstance().getStages().last()).intValue();

        for (int i=0; i < groundFloorColumns.size(); i++){
            Column c = (Column) groundFloorColumns.elementAt(i);
            Vector columnStack =
                Structure.getInstance().getColumnStack(c,lastDay);
            for (int j = 0; j < columnStack.size(); j++){
                Column c1 = (Column) columnStack.elementAt(j);
                for(int k=0; k < numRows; k++){
                    Column c2 = dialog.getColumn(k);
                    if (c1 == c2){
                        dialog.moveRow(k,k,rowPos);
                        rowPos++;
                        break;
                    }
                }
            }
        }

        dialog.pack();
        dialog.setVisible(true);
    }

  /** Inner class for the menus */
  private class AnalyserMenuBar extends JMenuBar{
    JMenu menu = null;
    JMenuItem newMenuItem = null;
    JMenuItem openMenuItem = null;
    JMenuItem saveMenuItem = null;
    JMenuItem saveAsMenuItem = null;
    JMenuItem exitMenuItem = null;
    JMenuItem newElementMenuItem = null;
    JMenuItem editElementMenuItem = null;
    JMenuItem deleteElementMenuItem = null;
    JMenuItem addFramingMenuItem = null;
    JMenuItem materialMenuItem = null;
```

```
    JMenuItem sectionMenuItem = null;
    JMenuItem editLoadsMenuItem = null;


    public AnalyserMenuBar(){
      setBackground(new Color(255,255,255,80));
      addFileMenu();
      addElementMenu();
      addLoadMenu();
    }

    /**Add the file menu to this menu bar */
    private void addFileMenu(){
      menu = new JMenu("File");
      menu.setMnemonic(KeyEvent.VK_F);
      add(menu);

      newMenuItem = new JMenuItem("New ...",KeyEvent.VK_N);
      newMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,
ActionEvent.ALT_MASK));
      newMenuItem.addActionListener(myNewAction);
      menu.add(newMenuItem);

      openMenuItem = new JMenuItem("Open ...",KeyEvent.VK_O);
      openMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O,
ActionEvent.ALT_MASK));
      openMenuItem.addActionListener(myOpenAction);
      menu.add(openMenuItem);

      saveMenuItem = new JMenuItem("Save",KeyEvent.VK_S);
      saveMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
ActionEvent.ALT_MASK));
      saveMenuItem.addActionListener(mySaveAction);
      menu.add(saveMenuItem);

      saveAsMenuItem = new JMenuItem("Save As...",KeyEvent.VK_A);
      saveAsMenuItem.addActionListener(mySaveAsAction);
      menu.add(saveAsMenuItem);
      menu.addSeparator();
      exitMenuItem = new JMenuItem("Exit",KeyEvent.VK_X);
      exitMenuItem.addActionListener(myExitAction);
      exitMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_X,
ActionEvent.ALT_MASK));
      menu.add(exitMenuItem);
    }


    /**add the Element menu to this menu bar */
    private void addElementMenu(){
      menu = new JMenu("Element");
      menu.setMnemonic(KeyEvent.VK_E);
      add(menu);

      materialMenuItem = new JMenuItem("New Material ...",KeyEvent.VK_M);
      materialMenuItem.addActionListener(myNewMaterialAction);
      materialMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_M,
ActionEvent.ALT_MASK));
```

```
        menu.add(materialMenuItem);
        sectionMenuItem = new JMenuItem("New Section ...",KeyEvent.VK_S);
        sectionMenuItem.addActionListener(myNewSectionAction);
        sectionMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
ActionEvent.ALT_MASK));
        menu.add(sectionMenuItem);

        newElementMenuItem = new JMenuItem("New Element ...",KeyEvent.VK_N);
        newElementMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,
ActionEvent.ALT_MASK));
        newElementMenuItem.addActionListener(myNewElementAction);
        menu.add(newElementMenuItem);

        editElementMenuItem = new JMenuItem("Edit Element ...",KeyEvent.VK_E);
        editElementMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_E,
ActionEvent.ALT_MASK));
        editElementMenuItem.addActionListener(myEditElementAction);
        menu.add(editElementMenuItem);

        deleteElementMenuItem =
          new JMenuItem("Delete Element ...",KeyEvent.VK_D);

deleteElementMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_D,
ActionEvent.ALT_MASK));
        deleteElementMenuItem.addActionListener(myDeleteAction);
        menu.add(deleteElementMenuItem);

        addFramingMenuItem =
          new JMenuItem("Add Framing ...",KeyEvent.VK_F);
        addFramingMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F,
ActionEvent.ALT_MASK));
        addFramingMenuItem.addActionListener(myAddFramingAction);
        menu.add(addFramingMenuItem);
    }

    /**add the Load menu to this menu bar */
    private void addLoadMenu(){
      menu = new JMenu("Load");
      menu.setMnemonic(KeyEvent.VK_L);
      add(menu);

      editLoadsMenuItem = new JMenuItem("Edit Loads...",KeyEvent.VK_E);
      editLoadsMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_E,
ActionEvent.ALT_MASK));
      editLoadsMenuItem.addActionListener(myEditLoadsAction);
      menu.add(editLoadsMenuItem);
    }

    /**Set the enabled states of the menu items */
    public void refresh(){
      if ((Structure.getInstance().getMaterials().size() > 0) &&
          (Structure.getInstance().getSections().size() > 0))
        newElementMenuItem.setEnabled(true);
      else
        newElementMenuItem.setEnabled(false);

      if (Structure.getInstance().getAllElements().isEmpty())
```

```
            addFramingMenuItem.setEnabled(false);
        else
            addFramingMenuItem.setEnabled(true);



        if (myElementsPane.getSelectedIndex() == -1){
          deleteElementMenuItem.setEnabled(false);
          editElementMenuItem.setEnabled(false);
          editLoadsMenuItem.setEnabled(false);
          return;
        }
        deleteElementMenuItem.setEnabled(true);
        editElementMenuItem.setEnabled(true);
        editLoadsMenuItem.setEnabled(true);
      }

    }


    /** Inner class for the elements pane. This pane displays the elements in
the
     * structure.
     */
    private class ElementsPane extends JPanel{
      /**The list to display the elements */
      JList myJlist = null;
      DefaultListModel myElements = null;

      /**Constructor */
      public ElementsPane(){
        myElements = new DefaultListModel();

        myJlist = new JList(myElements) {
            public String getToolTipText(MouseEvent e) {
              int index = locationToIndex(e.getPoint());
              if (-1 < index) {
                StructuralElement elt =
                  (StructuralElement) getModel().getElementAt(index);
                return elt.getToolTipText();
              } else {
                return null;
              }
            }
          };
        myJlist.setToolTipText("");
        myJlist.addListSelectionListener(new MyListSelectionListener());
        myJlist.getModel().addListDataListener(new MyListDataListener());
        myJlist.addKeyListener(new MyKeyListener());
        myJlist.addMouseListener(new ElementsListListener());
        add(new JScrollPane(myJlist));
      }

      /** Add an element to the list */
      public void addElement(StructuralElement elt){
        myElements.addElement(elt);
      }
```

```
/** Clear the list*/
public void clear(){
  myElements.clear();
}

/** Remove the selected element*/
public void removeSelectedElement(){
  int index = getSelectedIndex();
  if (index != -1)
    myElements.removeElementAt(index);
}

/** Get the index of the selected element in the list (return -1
    if no element is selected */
public int getSelectedIndex(){
  return myJlist.getSelectedIndex();
}

/** Get the element that is selected. Return null if no element is
    selected.*/
public StructuralElement getSelection(){
  int index = getSelectedIndex();
  if (index == -1)
    return null;
  else{
    Object elt = myElements.getElementAt(index);
    Assert.assert(elt instanceof StructuralElement);
    return (StructuralElement) elt;
  }
}

/**
 * Inner class --- list selection listener. Changes the button
 * states according to the selection of the element in the function
 * list.
 */
private class MyListSelectionListener implements ListSelectionListener{
  public void valueChanged (ListSelectionEvent evt) {
    //This method is called twice --- due to a mouse click *and*
    //mouse release.  Hence we only refresh for the mouse release
    if(!evt.getValueIsAdjusting()){
      SwingUtilities.invokeLater(refresh);
    }
  }
}

/**
 * MouseListener innerclass to handle mouse clicks on the list of items
 */
class ElementsListListener extends MouseAdapter{
  public void mouseClicked(MouseEvent me) {
    if (me.getClickCount() == 2)
      myEditLoadsAction.actionPerformed(null);
  }
}
```

```
    /**
     * Inner class --- list data listener. Refreshes the button states.
     */
    class MyListDataListener implements ListDataListener {
      public void contentsChanged(ListDataEvent e) {
        SwingUtilities.invokeLater(refresh);
      }
      public void intervalAdded(ListDataEvent e) {
        SwingUtilities.invokeLater(refresh);
      }
      public void intervalRemoved(ListDataEvent e) {
        SwingUtilities.invokeLater(refresh);
      }
    }

    /**
     * Inner class --- key event listener. Handles key event typed
     * into the list pane.
     */
    class MyKeyListener extends KeyAdapter {
      /** Handle the key typed event from the function pane. */
      public void keyPressed(KeyEvent e) {
        //if the delete key is pressed then delete the element
        if((e.getKeyCode() == KeyEvent.VK_DELETE) &&
           myElementsPane.getSelection() != null)
          myDeleteAction.actionPerformed(null);
      }
    }
  }

  /** Inner class for the buttons pane. This pane displays the Add
   *  and  Delete buttons.
   */
  private class ButtonsPane extends JPanel{
     /** Calculate shrinkage button */
    JButton myCalcShrinkageBtn = null;
    JButton myShowFramingShrinkageBtn = null;
    JButton myShowNonFramingShrinkageBtn = null;

    /**Constructor */
    public ButtonsPane(){
      myShowFramingShrinkageBtn = new JButton("<html>Show Framing<br>
Shrinkages</html>");

myShowFramingShrinkageBtn.addActionListener(myShowFramingShrinkageAction);
      myShowFramingShrinkageBtn.setToolTipText("Click to show the previously
calculated shrinkages (when framing was used in the calculations).");
      myShowFramingShrinkageBtn.setEnabled(false);

      myShowNonFramingShrinkageBtn = new JButton("<html>Show Non Framing<br>
Shrinkages</html>");

myShowNonFramingShrinkageBtn.addActionListener(myShowNonFramingShrinkageAction
);
```

```
      myShowNonFramingShrinkageBtn.setToolTipText("Click to show the
previously calculated shrinkages (when framing was not used in the
calculations).");
      myShowNonFramingShrinkageBtn.setEnabled(false);

      myCalcShrinkageBtn = new JButton("<html>Calculate New<br>
Shrinkages</html");
      myCalcShrinkageBtn.addActionListener(myCalcShrinkageAction);
      myCalcShrinkageBtn.setToolTipText("Click to calculate new shrinkages.");
      myCalcShrinkageBtn.setEnabled(false);

      // add the buttons
      setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
      setBorder(myPaneEdgeBdr);
      add(myShowFramingShrinkageBtn);
      add(Box.createRigidArea(new Dimension(10, 0)));
      add(myShowNonFramingShrinkageBtn);
      add(Box.createRigidArea(new Dimension(10, 0)));
      add(myCalcShrinkageBtn);
    }

    /**
     * Refresh the presentation state of the buttons.
     */
    public void refresh() {
      //set the calcShrinkage button to enabled iff there is at least one
      //column in the structure
      boolean canCalcShrinkage = false;
      Vector allElements = Structure.getInstance().getAllElements();
      for (int i = 0; i < allElements.size(); i++){
        if (allElements.elementAt(i) instanceof Column){
          canCalcShrinkage = true;
          break;
        }
      }
      myCalcShrinkageBtn.setEnabled(canCalcShrinkage);
      //set the show shrinkages btn enabled if there are shrinkages calculated
      boolean enable = Structure.getInstance().getShrinkageDays(false).size()
>0;

      myShowNonFramingShrinkageBtn.setEnabled(enable);
      enable = Structure.getInstance().getShrinkageDays(true).size() >0;
      myShowFramingShrinkageBtn.setEnabled(enable);
    }
  }

  private void save(){
    if (myFilePath == null){
      FileDialog fd = new FileDialog(this, "Save As...", FileDialog.SAVE );
      fd.show();
      if(fd.getDirectory() != null && fd.getFile() != null)
        myFilePath = new String(fd.getDirectory() + fd.getFile());
    }
    if (myFilePath != null){
      try{
        //  Create a stream for writing.
        FileOutputStream fos = new FileOutputStream(myFilePath);
```

```
        //Create an object that can write to that file.
        ObjectOutputStream outStream = new ObjectOutputStream(fos);
        outStream.writeObject(Structure.getInstance());
        outStream.flush();
      }
      catch(FileNotFoundException e){
        System.err.println("File not found exception.");
      }
      catch ( IOException e ) {
        System.err.println( "IO exception " + e.getMessage() );
      }
    }
  }

  /**
   * Inner class that defines the actions performed to add to the list
   */
  private class NewElementAction extends AbstractAction {
    public void actionPerformed(ActionEvent evt) {
      JDialog dialog = new NewElementDialog(AnalyserFrame.this);
      dialog.pack();
      dialog.setVisible(true);
    }
  }

  /**
   * Inner class that defines the actions performed to edit an element
   * from the list
   */
  private class EditElementAction extends AbstractAction {
    public void actionPerformed(ActionEvent evt) {
      StructuralElement elt = myElementsPane.getSelection();
      JDialog dialog = new EditElementDialog(AnalyserFrame.this, elt);
      dialog.pack();
      dialog.setVisible(true);
    }
  }


  /**
   * Inner class that defines the actions performed to delete from the list
   */
  private class DeleteAction extends AbstractAction {
    public void actionPerformed(ActionEvent evt) {
      StructuralElement elt = myElementsPane.getSelection();
      if(Structure.getInstance().canRemoveElement(elt)){
        Structure.getInstance().removeElement(elt);
        myElementsPane.removeSelectedElement();
      }
      else
        JOptionPane.showMessageDialog(AnalyserFrame.this,
                                      "You cannot delete this element, since
it supports other structural elements.",
                                      "Error",
                                      JOptionPane.ERROR_MESSAGE);
    }
  }
```

```
/**
 * Inner class that defines the actions performed to calc shrinkage
 */
private class EditLoadsAction extends AbstractAction {
  public void actionPerformed(ActionEvent evt) {
    JDialog dialog = new LoadsDialog(AnalyserFrame.this,
                                     myElementsPane.getSelection());
    dialog.pack();
    dialog.setVisible(true);
  }
}

/**
 * Inner class that defines the new material action performed
 */
private class NewMaterialAction extends AbstractAction {
  public void actionPerformed(ActionEvent evt) {
    JDialog dialog = new NewMaterialDialog(AnalyserFrame.this);
    dialog.pack();
    dialog.setVisible(true);
    //refresh the buttons and menu items
    SwingUtilities.invokeLater(refresh);
  }
}

/**
 * Inner class that defines the new section action performed
 */
private class NewSectionAction extends AbstractAction {
  public void actionPerformed(ActionEvent evt) {
    JDialog dialog = new NewSectionDialog(AnalyserFrame.this);
    dialog.pack();
    dialog.setVisible(true);
    //refresh the buttons and menu items
    SwingUtilities.invokeLater(refresh);
  }
}

/**
 * Inner class that defines the actions performed to calc shrinkage
 */
private class CalcShrinkageAction extends AbstractAction {
  public void actionPerformed(ActionEvent evt) {
    CalcShrinkageDialog dialog =
      new CalcShrinkageDialog(AnalyserFrame.this);
    dialog.pack();
    dialog.setVisible(true);
  }
}

/**
 * Inner class that defines the actions performed to show shrinkages
 */
private class ShowFramingShrinkageAction extends AbstractAction {
  public void actionPerformed(ActionEvent evt) {
      displayShrinkages(AnalyserFrame.this,true);
```

```
  }
}

/**
 * Inner class that defines the actions performed to show shrinkages
 */
private class ShowNonFramingShrinkageAction extends AbstractAction {
  public void actionPerformed(ActionEvent evt) {
      displayShrinkages(AnalyserFrame.this,false);
  }
}

/**
 * Inner class that defines the actions performed to calc shrinkage
 */
  private class AddFramingAction extends AbstractAction {
      public void actionPerformed(ActionEvent evt) {
          AddFramingDialog dialog =
              new AddFramingDialog(AnalyserFrame.this);
          dialog.pack();
          dialog.setVisible(true);
      }
  }


/**The action that is performed to open a file */
private class NewAction extends AbstractAction {
  public void actionPerformed(ActionEvent evt) {
    /** should we create a new structure */
    boolean toCreateNew = true;
    if (Structure.getInstance().getStructureChanged()){
      int selectedValue =
        JOptionPane.showConfirmDialog(AnalyserFrame.this,
                                      "Do you wish to save the changes to" +
                                      myFilePath,
                                      "Save?",
                                      JOptionPane.YES_NO_CANCEL_OPTION,
                                      JOptionPane.INFORMATION_MESSAGE);
      if (selectedValue == JOptionPane.YES_OPTION){
        save();
      }
      if (selectedValue == JOptionPane.CANCEL_OPTION)
        toCreateNew = false;
    }

    if (toCreateNew){
      clear();
      Structure.setInstance(new Structure());
      //refresh the buttons and menu items
      SwingUtilities.invokeLater(refresh);
    }
  }
}


/**The action that is performed to open a file */
private class OpenAction extends AbstractAction {
```

```
    public void actionPerformed(ActionEvent evt) {
      /** should we try to open another file */
      boolean toOpen = true;
      if (Structure.getInstance().getStructureChanged()){
        int selectedValue =
          JOptionPane.showConfirmDialog(AnalyserFrame.this,
                                        "Do you wish to save the changes to" +
                                        myFilePath,
                                        "Save?",
                                        JOptionPane.YES_NO_CANCEL_OPTION,
                                        JOptionPane.INFORMATION_MESSAGE);
        if (selectedValue == JOptionPane.YES_OPTION){
          save();
        }
        if (selectedValue == JOptionPane.CANCEL_OPTION)
          toOpen = false;
      }

      if (toOpen){
        FileDialog fd = new FileDialog(AnalyserFrame.this, "Open...",
                                       FileDialog.LOAD);
        fd.show();
        if(fd.getDirectory() != null && fd.getFile() != null){
          try{
            String myOldFilePath = myFilePath;
            myFilePath = new String(fd.getDirectory() + fd.getFile());
            if (myFilePath.equals(myOldFilePath) &&
                Structure.getInstance().getStructureChanged()){
              int selectedValue =
                JOptionPane.showConfirmDialog(AnalyserFrame.this,
                                              "Do you wish to revert to
previously saved version?",
                                              "Open?",
                                              JOptionPane.YES_NO_OPTION,

JOptionPane.INFORMATION_MESSAGE);
              if (selectedValue == JOptionPane.NO_OPTION)
                return;
            }

            //Create a stream for reading.
            FileInputStream fis = new FileInputStream(myFilePath);
            //Create an object that can read from that file.
            ObjectInputStream inStream = new ObjectInputStream( fis );
            // Retrieve the structure
            Structure.setInstance((Structure) inStream.readObject());
            Structure.getInstance().setStructureChanged(false);
            Vector allElements = Structure.getInstance().getAllElements();
            clear();
            if (allElements != null){
              for (int i =0; i < allElements.size(); i++){
                addElement((StructuralElement) allElements.elementAt(i));
              }
            }
            //refresh the buttons and menu items
            SwingUtilities.invokeLater(refresh);
          }
```

```
            catch(FileNotFoundException e){
              System.err.println("File not found exception.");
            }
            catch (ClassNotFoundException e) {
              System.err.println("ClassNotFoundException...");
            }
            catch(OptionalDataException e){
              System.err.println("OptionalDataException.");
            }
            catch ( IOException e ) {
              System.err.println( "IO exception " + e.getMessage() );
            }
          }
        }
      }
    }

    private class SaveAction extends AbstractAction {
      public void actionPerformed(ActionEvent evt) {
        save();
        Structure.getInstance().setStructureChanged(false);
      }
    }

    private class SaveAsAction extends AbstractAction {
      public void actionPerformed(ActionEvent evt) {
        FileDialog fd = new FileDialog(AnalyserFrame.this,
                                       "Save As...", FileDialog.SAVE );
        fd.show();
        if(fd.getDirectory() != null && fd.getFile() != null){
          myFilePath = new String(fd.getDirectory() + fd.getFile() );
          save();
        }
      }
    }

    private class ExitAction extends AbstractAction {
      public void actionPerformed(ActionEvent evt) {
        if (Structure.getInstance().getStructureChanged()){
          int selectedValue =
            JOptionPane.showConfirmDialog(AnalyserFrame.this,
                                          "Do you wish to save your changes?",
                                          "Save?",
                                          JOptionPane.YES_NO_CANCEL_OPTION,
                                          JOptionPane.INFORMATION_MESSAGE);
          if (selectedValue == JOptionPane.YES_OPTION){
            save();
            System.exit(0);
          }
          else if (selectedValue == JOptionPane.NO_OPTION)
            System.exit(0);
        }
        else
          System.exit(0);
      }
    }
}
```

```
/*
 * @(#)Assert.java
 *
 * History:  Nov 2001: Written by M. Gardner and G. Lewis
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;
import java.io.*;

/**
 * A simple assertion mechanism. Assertions will be introduced in Java
 * 1.4 (aka Merlin), but for now, we use this class.
 *
 */
public class Assert
{
  /** Get a stack trace*/
  private static String getStackTrace(){
    Throwable t = new Throwable();
    ByteArrayOutputStream os = new ByteArrayOutputStream();
    PrintStream ps = new PrintStream(os);
    t.printStackTrace(ps);
    return os.toString();
  }

  /** Check a precondition. If it fails then a message and stack trace
   * is printed to <code>System.err</code>.
   *@param condition the precondition to check
   *@param message the message to print if the condition is not satisified */
  public static void preCondition(boolean condition, String message){
    if (!condition){
      if (message != null)
      System.err.println("Precondition: " + message + " failed at:");
      else
      System.err.println("Precondition failed at:");
      System.err.println(getStackTrace());
      System.exit(1);
    }
  }

  /** Check a precondition. If it fails then a stack trace
   * is printed to <code>System.err</code>.
   *@param condition the precondition to check
   */
  public static void preCondition(boolean condition){
    preCondition(condition,null);
  }


  /** Check an assertion. If it fails then a message and stack trace
   * is printed to <code>System.err</code>.
   *@param condition the condition to check
   *@param message the message to print if the condition is not satisified
   */
```

```
  public static void assert(boolean condition, String message){
    if (!condition) {
      if (message != null)
      System.err.println("Assert: " + message + " failed at:");
      else
      System.err.println("Assert failed at:");
      System.err.println(getStackTrace());
      System.exit(1);
    }
  }

  /** Check an assertion. If it fails then a stack trace
   * is printed to <code>System.err</code>.
   *@param condition the condition to check
   */
  public static void assert(boolean condition){
    assert(condition,null);
  }

  /** Check a postcondition. If it fails then a message and stack trace
   * is printed to <code>System.err</code>.
   *@param condition the postcondition to check
   *@param message the message to print if the condition is not satisified
   */
  public static void postCondition(boolean condition, String message)
  {
    if (!condition){
      if (message != null)
      System.err.println("Postcondition: " + message + " failed at:");
      else
      System.err.println("Postcondition failed at:");
      System.err.println(getStackTrace());
      System.exit(1);
    }
  }


  /** Check a postcondition. If it fails then a stack trace is printed
   * to <code>System.err</code>.
   *@param condition the postcondition to check
   */
   public static void postCondition(boolean condition){
    postCondition(condition,null);
  }

  /** Exit the program printing a stack trace on exit. This method is
   *  typically used in a stub method that is not implemented yet
   */
  public static void abruptExit(){
    System.err.println("Abrupt Exit");
    System.err.println(getStackTrace());
    System.exit(1);
  }
}
```

```
/*
 * @(#) CalcShrinkageDialog.java
 *
 * History:  Nov 2001: Written by M. Gardner and G. Lewis
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

import javax.swing.*;
import javax.swing.border.*;
import javax.accessibility.*;

import java.util.*;
import java.awt.*;
import java.awt.event.*;

import structuralAnalyser.structuralElement.*;
import structuralAnalyser.material.*;

/**Dialog to calculate the shrinkage.*/
public class CalcShrinkageDialog extends JDialog implements WindowListener{

  /** The pane that displays the calculate and done buttons*/
  JPanel myCalculateDonePane = null;
  /** Done Action */
  Action myDoneAction = new DoneAction();
  /** Calculate Action */
  Action myCalculateAction = new CalculateAction();
  /**A border that puts 10 extra pixels around a pane.*/
  Border myPaneEdgeBdr = BorderFactory.createEmptyBorder(10,10,10,10);
  /**The day the column was constructed */
  int day;
  JLabel dayLabel = null;
  JTextField dayField = null;
  /**The x position of the column */
  int xCoord;
  JLabel xCoordLabel = null;
  JTextField xCoordField = null;
  JRadioButton xCoordBtn = null;
  /**The y position of the column */
  int yCoord;
  JLabel yCoordLabel = null;
  JTextField yCoordField = null;
  JRadioButton yCoordBtn = null;
  /**The z position of the column */
  int zCoord;
  JLabel zCoordLabel = null;
  JTextField zCoordField = null;
  JRadioButton zCoordBtn = null;
    /**Include framing in calculation? */
    JLabel includeFramingLabel = null;
    JCheckBox includeFramingBtn = null;

 /**The parent frame */
```

```
AnalyserFrame myParent = null;

/** The content pane */
private Container myContentPane = getContentPane();

public CalcShrinkageDialog(AnalyserFrame parent) {
  super(parent,"Calc Delta Dialog",true);
   myParent = parent;
   addWindowListener(this);
  myContentPane.setLayout(new GridLayout(0,1));

  JPanel dayPanel = new JPanel();
  dayLabel = new JLabel("Day");
  dayField = new JTextField(4);
  dayPanel.add(dayLabel);
  dayPanel.add(dayField);

  JPanel xCoordPanel = new JPanel();
  xCoordLabel = new JLabel("X Coordinate");
  xCoordField = new JTextField(4);
  xCoordPanel.add(xCoordLabel);
  xCoordPanel.add(xCoordField);
  xCoordBtn = new JRadioButton("All X coordinates");
  xCoordBtn.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e)
    {
      xCoordField.setText("");
      xCoordField.setEditable(!xCoordBtn.isSelected());
    }
    });
  xCoordPanel.add(xCoordBtn);



  JPanel yCoordPanel = new JPanel();
  yCoordLabel = new JLabel("Y Coordinate");
  yCoordField = new JTextField(4);
  yCoordPanel.add(yCoordLabel);
  yCoordPanel.add(yCoordField);
  yCoordBtn = new JRadioButton("All Y coordinates");
  yCoordBtn.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e)
    {
      yCoordField.setText("");
      yCoordField.setEditable(!yCoordBtn.isSelected());
    }
    });
  yCoordPanel.add(yCoordBtn);



  JPanel zCoordPanel = new JPanel();
  zCoordLabel = new JLabel("Z Coordinate");
  zCoordField = new JTextField(4);
  zCoordPanel.add(zCoordLabel);
  zCoordPanel.add(zCoordField);
  zCoordBtn = new JRadioButton("All Z coordinates");
```

```
    zCoordBtn.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e)
      {
        zCoordField.setText("");
        zCoordField.setEditable(!zCoordBtn.isSelected());
      }
      });
    zCoordPanel.add(zCoordBtn);

    JPanel includeFramingPanel = new JPanel();
    includeFramingLabel = new JLabel("Include Framing in calculation");
    includeFramingBtn = new JCheckBox();
    includeFramingPanel.add(includeFramingLabel);
    includeFramingPanel.add(includeFramingBtn);


    myContentPane.add(dayPanel);
    myContentPane.add(xCoordPanel);
    myContentPane.add(yCoordPanel);
    myContentPane.add(zCoordPanel);
    myContentPane.add(includeFramingPanel);

    myCalculateDonePane = new CalculateDonePane();
    myContentPane.add(myCalculateDonePane);
  }

    /** handle the closing event */
    public void windowClosing(WindowEvent e) {
        setVisible(false);
      myParent.refresh();
    }

    public void windowClosed(WindowEvent e) {}

    public void windowOpened(WindowEvent e) {}

    public void windowIconified(WindowEvent e) {}

    public void windowDeiconified(WindowEvent e) {}

    public void windowActivated(WindowEvent e) {}

    public void windowDeactivated(WindowEvent e) {}

    /**Parse the input typed by the user. */
    public boolean parseInput(){
      try{
      day = Integer.parseInt(dayField.getText());
      if (day < 0)
        throw new NumberFormatException();
      }
      catch (NumberFormatException e){
      JOptionPane.showMessageDialog(this,
                                    "The day is not valid",
                                "Error",
                                    JOptionPane.ERROR_MESSAGE);
      return false;
```

```
      }

      if (!xCoordBtn.isSelected()){
      try{
        xCoord = Integer.parseInt(xCoordField.getText());
        if (xCoord < 0)
          throw new NumberFormatException();
      }
      catch (NumberFormatException e){
        JOptionPane.showMessageDialog(this,
                                "The x-coordinate is not valid",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
        return false;
      }
      }

      if (!yCoordBtn.isSelected()){
      try{
        yCoord = Integer.parseInt(yCoordField.getText());
        if (yCoord < 0)
          throw new NumberFormatException();
      }
      catch (NumberFormatException e){
        JOptionPane.showMessageDialog(this,
                                "The y-coordinate is not valid",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
        return false;
      }
      }

      if (!zCoordBtn.isSelected()){
      try{
        zCoord = Integer.parseInt(zCoordField.getText());
        if (zCoord < 0)
          throw new NumberFormatException();
      }
      catch (NumberFormatException e){
        JOptionPane.showMessageDialog(this,
                                "The z-coordinate is not valid",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
        return false;
      }
      }
      return true;
    }
/**
   * The calculate/done pane. Displays the calculate and done
   * buttons.
   */
  private class CalculateDonePane extends JPanel{
    /** Done Button */
    JButton myDoneBtn = null;
    /** Calculate Button */
    JButton myCalculateBtn = null;
```

```
    /**Constructor*/
    public CalculateDonePane(){
      // create the buttons and register the actions
      myCalculateBtn = new JButton("Calculate");
      myCalculateBtn.addActionListener(myCalculateAction);
      myCalculateBtn.setToolTipText("Click to calculate.");
      myDoneBtn = new JButton("Cancel");
      myDoneBtn.addActionListener(myDoneAction);
      myDoneBtn.setToolTipText("Click to cancel.");

      //add the buttons to the pane
      BoxLayout layout = new BoxLayout(this, BoxLayout.X_AXIS);
      setLayout(layout);
      setBorder(myPaneEdgeBdr);
      add(Box.createHorizontalGlue());
      add(myDoneBtn);
      add(Box.createRigidArea(new Dimension(10, 0)));
      add(myCalculateBtn);
    }
  }
    /** Add the shrinkages to the structure, and display all shrinkages
     * @param columns the columns to calculate the shrinkages for
     * @param framing should framing be included in the calculation?
     */
  // private void addShrinkages(Vector columns, boolean framing) {
  //try{
  // Vector deltas = null;
  // if (framing)
  //deltas = FramingAlgorithm.getInstance().calcColumnDelta(columns,day);
  // else
  //deltas = NonFramingAlgorithm.getInstance().calcColumnDelta(columns,day);
          //add the shrinkages to the structure
  // Assert.assert(columns.size() == deltas.size());
  // for(int i =0; i < deltas.size(); i++){
  //Column col = (Column) columns.elementAt(i);
  //double shrinkage = ((Double) deltas.elementAt(i)).doubleValue();
  //Structure.getInstance().addShrinkage(day,col,shrinkage,framing);
  //}
  // AnalyserFrame.displayShrinkages(CalcShrinkageDialog.this,framing);
  //}
  //catch(InvalidParameterException e){
  //  System.err.println(e.getMessage());
  //}
  // }
 /** Add the shrinkages to the structure, and display all shrinkages(THIS IS
MY ATTEMPT)
     * @param columns the columns to calculate the shrinkages for
     * @param framing shold framing be included in the calculation?
     */

  private void addShrinkages(Vector columns, boolean framing) {
      try{
          Vector columnDeltas = null;
          if (framing)
            columnDeltas =
FramingAlgorithm.getInstance().calcColumnDelta(columns,day);
```

```
          else
            columnDeltas =
NonFramingAlgorithm.getInstance().calcColumnDelta(columns,day);
          //add the shrinkages to the structure
          Assert.assert(columns.size() == columnDeltas.size());
          for(int i =0; i < columnDeltas.size(); i++){
            Column col = (Column) columnDeltas.elementAt(i);
            double shrinkage = col.getTempDelta();
            //double shrinkage = ((Double) deltas.elementAt(i)).doubleValue();
            Structure.getInstance().addShrinkage(day,col,shrinkage,framing);
      }
          AnalyserFrame.displayShrinkages(CalcShrinkageDialog.this,framing);
      }
      catch(InvalidParameterException e){
          System.err.println(e.getMessage());
      }
    }
   /**
   * Inner class that defines the calculate action
   */
  private class CalculateAction extends AbstractAction {
    public void actionPerformed(ActionEvent evt) {
      //The columns to calculate shrinkages for
      Vector columns = new Vector();
      Vector allElements = Structure.getInstance().getAllElements();

      //find the columns, calculate and display the shrinkages
      try{
          if (parseInput()){
            //if calculating delta for a single column
            if (!xCoordBtn.isSelected() && !yCoordBtn.isSelected() &&
                !zCoordBtn.isSelected()){
                Coordinate columnCoordinate = new
Coordinate(xCoord,yCoord,zCoord);
                // find the column with the given coordinate
                Column column = null;
                for(int i = 0; i< allElements.size(); i++){
                  StructuralElement elt = (StructuralElement)
allElements.elementAt(i);
                  if ((elt instanceof Column) &&
                      (elt.getStartCoordinate().isEqual(columnCoordinate))){
                      column = (Column) elt;
                      break;
                  }
                }
                columns.addElement(column);
            }
            else if (!xCoordBtn.isSelected() && yCoordBtn.isSelected() &&
                   !zCoordBtn.isSelected()){
                float xPos = xCoord;
                float zPos = zCoord;
                // find the column with the given coordinate
                for(int i = 0; i< allElements.size(); i++){
                  StructuralElement elt = (StructuralElement)
                      allElements.elementAt(i);
                  Coordinate thisCoord = elt.getStartCoordinate();
                  float thisXPos = thisCoord.getXPosition();
```

```
        float thisZPos = thisCoord.getZPosition();

        if ((elt instanceof Column) && (thisXPos == xPos) &&
            (thisZPos == zPos))
            columns.addElement((Column) elt);
      }
    }
    else if (xCoordBtn.isSelected() &&!yCoordBtn.isSelected() &&
           !zCoordBtn.isSelected()){
      float yPos = yCoord;
      float zPos = zCoord;
      for(int i = 0; i< allElements.size(); i++){
        StructuralElement elt = (StructuralElement)
            allElements.elementAt(i);
        Coordinate thisCoord = elt.getStartCoordinate();
        float thisYPos = thisCoord.getYPosition();
        float thisZPos = thisCoord.getZPosition();
        if ((elt instanceof Column) && (thisYPos == yPos) &&
            (thisZPos == zPos))
            columns.addElement((Column) elt);
      }
    }
    else if (!xCoordBtn.isSelected() && !yCoordBtn.isSelected() &&
           zCoordBtn.isSelected()){
      float xPos = xCoord;
      float yPos = yCoord;
      for(int i = 0; i< allElements.size(); i++){
        StructuralElement elt = (StructuralElement)
            allElements.elementAt(i);
        Coordinate thisCoord = elt.getStartCoordinate();
        float thisXPos = thisCoord.getXPosition();
        float thisYPos = thisCoord.getYPosition();
        if ((elt instanceof Column) && (thisXPos == xPos) &&
            (thisYPos == yPos)){
            columns.addElement((Column) elt);
        }
      }
    }
    else if (xCoordBtn.isSelected() && yCoordBtn.isSelected() &&
           !zCoordBtn.isSelected()){
      float zPos = zCoord;
      for(int i = 0; i< allElements.size(); i++){
        StructuralElement elt = (StructuralElement)
            allElements.elementAt(i);
        Coordinate thisCoord = elt.getStartCoordinate();
        float thisZPos = thisCoord.getZPosition();
        if ((elt instanceof Column) && (thisZPos == zPos))
            columns.addElement((Column) elt);
      }
    }
    else if (!xCoordBtn.isSelected() && yCoordBtn.isSelected() &&
           zCoordBtn.isSelected()){
      float xPos = xCoord;
      for(int i = 0; i< allElements.size(); i++){
        StructuralElement elt = (StructuralElement)
            allElements.elementAt(i);
        Coordinate thisCoord = elt.getStartCoordinate();
```

```
                    float thisXPos = thisCoord.getXPosition();
                    if ((elt instanceof Column) && (thisXPos == xPos)){
                        columns.addElement((Column) elt);
                    }
                }
            }
            else if (xCoordBtn.isSelected() && !yCoordBtn.isSelected() &&
                    zCoordBtn.isSelected()){
                float yPos = yCoord;
                for(int i = 0; i< allElements.size(); i++){
                    StructuralElement elt =
                        (StructuralElement) allElements.elementAt(i);
                    Coordinate thisCoord = elt.getStartCoordinate();
                    float thisYPos = thisCoord.getYPosition();
                    if ((elt instanceof Column) && (thisYPos == yPos))
                        columns.addElement((Column) elt);
                }
            }
            else if (xCoordBtn.isSelected() && yCoordBtn.isSelected() &&
                    zCoordBtn.isSelected()){
                for(int i = 0; i< allElements.size(); i++){
                    StructuralElement elt = (StructuralElement)
                        allElements.elementAt(i);
                    if (elt instanceof Column)
                        columns.addElement((Column) elt);
                }
            }


            if (columns.size() == 0)
                JOptionPane.showMessageDialog(CalcShrinkageDialog.this,
                                    "Could not find any columns with these
coordinates.");
            else{
                addShrinkages(columns,includeFramingBtn.isSelected());

            }
          }
        }
        catch(InvalidParameterException e){
            System.err.println(e.getMessage());
        }
      }
    }

    /**
     * Inner class that defines the done action
     */
    private class DoneAction extends AbstractAction {
      public void actionPerformed(ActionEvent evt) {
        setVisible(false);
        myParent.refresh();
      }
    }

}
```

```
/*
 * @(#)Column.java
 *
 * History:  1999-2000: Written by M. Gardner and C. Dalton.
 *           Oct 2001:  G.Lewis added comments, deleted obsolete code,
 *                and changed variable names.
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser.structuralElement;

import java.io.*;
import structuralAnalyser.*;
import structuralAnalyser.material.*;

/** A Column structural element */
public class Column extends StructuralElement implements Serializable
{
  /**The length of the column */
  protected float myLength;
  protected double myTempDelta;
  protected double myTempDeltaExist;
  protected double myTempDeltaNew;
  protected double myTempCreep;
  protected double myTempCreepPrevious;
  protected Section mySection;
  protected boolean myConstructed;

  /**Construct a new column
   * @param length the length of the column
   */
  public Column(int id, Material material, float length, Section section)
    throws InvalidParameterException
  {
    super(id,material);
    setWeight(calculateWeight(material,length,section));
    setLength(length);
    setSection(section);
    myTempDelta = 0;
    myTempDeltaExist = 0;
    myTempDeltaNew = 0;
    myTempCreep = 0;
    myTempCreepPrevious = 0;
    myConstructed = false;
  }

  /**Get the length of this column */
  public float getLength()
  {
    return myLength;
  }

  /**Set the length of this column */
  public void setLength(float length) throws InvalidParameterException
  {
```

```
    if (length <= 0)
      throw new InvalidParameterException("The length must be greater than
0.");
    myLength = length;
  }

  /**Get the section of this column */
  public Section getSection()
  {
    return mySection;
  }

  /**Set the section of this column */
  public void setSection(Section section) throws InvalidParameterException
  {
    if (section == null)
      throw new InvalidParameterException("The section is null.");
    mySection = section;
  }

  /**Calculate the weight of the column*/
  private float calculateWeight(Material material, float length, Section
section){
    String kind = material.getKind();
    float weight = 1;
    double calculatedWeight = 1;
    float density = 1;
    double gravity = 0.00981;

    if (material instanceof ACIConcrete){
      ACIConcrete thisMaterial= (ACIConcrete)material;
       density = thisMaterial.getDensity();
    }

    else if  (material instanceof B3Concrete){
      B3Concrete thisMaterial= (B3Concrete)material;
       density = thisMaterial.getDensity();
    }

    else if  (material instanceof Metal){
     Metal thisMaterial= (Metal)material;
       density = thisMaterial.getDensity();
    }



    float area = section.getArea();
    calculatedWeight = density*area*length*9.81/1000;
    weight = Float.valueOf(Double.toString(calculatedWeight)).floatValue();


    return weight;
  }


  /**Get the temporary delta  of this column */
  public double getTempDelta()
```

```
  {
    return myTempDelta;
  }

  /**Set the temporary delta of this column */
  public void setTempDelta(double delta)
  {

    myTempDelta = delta;

  }


 /**Get the temporary creep for a stage  of this column */
  public double getTempCreep()
  {
    return myTempCreep;
  }

  /**Set the temporary creep for a stage  of this column */
  public void setTempCreep(double delta)
  {

    myTempCreep = delta;

  }


/**Get the temporary previous  creep for a stage  of this column */
  public double getTempCreepPrevious()
  {
    return myTempCreepPrevious;
  }

  /**Set the temporary previous creep for a stage  of this column */
  public void setTempCreepPrevious(double delta)
  {

    myTempCreepPrevious = delta;

  }


  /**Get the temporary constructed type for this column */
  public boolean getConstructed()
  {
    return myConstructed ;
  }

  /**Set the temp constructed type for this column */
  public void setConstructed(boolean b)
  {

    myConstructed = b;

  }
```

```
  /**Get the temporary delta Exist  of this column */
  public double getTempDeltaExist()
  {
    return myTempDeltaExist;
  }

  /**Set the temporary delta Exist of this column */
  public void setTempDeltaExist(double delta)
  {

    myTempDeltaExist = delta;

  }

  /**Get the temporary delta  New of this column */
  public double getTempDeltaNew(){
    return myTempDeltaNew;
  }

  /**Set the temporary delta New of this column */
  public void setTempDeltaNew(double delta)
  {

    myTempDeltaNew = delta;

  }

  /** Get the tooltip text */
  public String getToolTipText(){
    return "Double click on this column to add loads to it";
  }

  /**Return a string representation of this object */
  public String toString(){
    return "Column " + getId() + ": day = " + getDayConstructed() + "
coordinate = " +
      getStartCoordinate() + " length = " + getLength();
  }
}
```

```
/*
 * @(#)Concrete.java
 *
 * History:  1999-2000: Written by M Gardner and C. Dalton
 *           Oct 2001:  G.Lewis added comments, deleted obsolete code,
 *                and changed variable names.
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser.material;

import java.io.*;
import structuralAnalyser.*;

/** Abstract class representing rete. This class is sublclass for B3 concrete,
ACI concrete, etc.*/
public abstract class Concrete extends Material
{
  protected float myConc28day;
  protected float myDensity;
  protected float myElasticMod;
  protected float myBasicShrinkage;
  protected float myHumidity;
  protected float myMetalReinforcingArea;
  protected String myCementType;
  protected float myCementContent;
  protected float myWaterCementRatio;
  protected Metal myAddedMetal;

  /**Constructor*/
  public Concrete(String id, String kind, float conc28day, float density,
              float elasticMod, float basicShrinkage,
              float humidity, float metalReinforcingArea,
              String cementType, float cementContent,
              float waterCementRatio, Metal addedMetal)
              throws InvalidParameterException
  {
    super(id, kind);
    setConc28day(conc28day);
    setDensity(density);
    setElasticMod(elasticMod);
    setBasicShrinkage(basicShrinkage);
    setHumidity(humidity);
    setMetalReinforcingArea(metalReinforcingArea);
    setCementType(cementType);
    setCementContent(cementContent);
    setWaterCementRatio(waterCementRatio);
    setAddedMetal(addedMetal);
  }
  public float getConc28day()
  {
    return myConc28day;
  }
  public void setConc28day(float conc28day) throws InvalidParameterException
  {
```

```
    if (conc28day <= 10)
      throw new InvalidParameterException("Conc 28 day must be greater than
10.");
    myConc28day = conc28day;
  }
  public float getDensity()
  {
    return myDensity;
  }
  public void setDensity(float density) throws InvalidParameterException
  {
    if (density <= 1000)
      throw new InvalidParameterException("Density must be greater than
1000.");
    myDensity = density;
  }
  public float getElasticMod()
  {
    return myElasticMod;
  }
  public void setElasticMod(float elasticMod) throws InvalidParameterException
  {
    if (elasticMod < 0)
      throw new InvalidParameterException("Elastic Mod must be greater than
0.");
    myElasticMod = elasticMod;
  }
  public float getBasicShrinkage()
  {
    return myBasicShrinkage;
  }
  public void setBasicShrinkage(float basicShrinkage) throws
InvalidParameterException
  {
    if (basicShrinkage < 0)
      throw new InvalidParameterException("Thermal exp must be greater than
0.");
    myBasicShrinkage = basicShrinkage;
  }
  public float getHumidity()
  {
    return myHumidity;
  }
  public void setHumidity(float humidity) throws InvalidParameterException
  {
    if ( humidity < 0)
      throw new InvalidParameterException("Humidity must be greater than 0.");
    myHumidity = humidity;
  }
  public float getMetalReinforcingArea()
  {
    return myMetalReinforcingArea;
  }
  public void setMetalReinforcingArea(float metalReinforcingArea)
    throws InvalidParameterException
  {
    if (metalReinforcingArea < 0)
```

```
      throw new InvalidParameterException("Metal reinforcing area must be
greater than or equal to 0.");
      myMetalReinforcingArea = metalReinforcingArea;
  }
  public String getCementType()
  {
    return myCementType;
  }
  public void setCementType(String cementType) throws
InvalidParameterException
  {
    if (cementType == null)//THIS SHOULD BE A CHAR VALUE
      throw new InvalidParameterException("Cement type must be greater than or
equal to 0.");
      myCementType = cementType;
  }

  public float getCementContent()
  {
    return myCementContent;
  }

  public void setCementContent(float cementContent) throws
InvalidParameterException
  {
    if (cementContent < 0)
      throw new InvalidParameterException("Cement content must be greater than
or equal to 0.");
      myCementContent = cementContent;
  }

  public float getWaterCementRatio()
  {
    return myWaterCementRatio;
  }

  public void setWaterCementRatio(float waterCementRatio) throws
InvalidParameterException
  {
    if (waterCementRatio < 0)
      throw new InvalidParameterException("Water cement ratio must be greater
than or equal to 0.");
     myWaterCementRatio = waterCementRatio;
  }

  public Metal getAddedMetal()
  {
    return myAddedMetal;
  }

  public void setAddedMetal(Metal addedMetal) throws InvalidParameterException
  {
    if (addedMetal == null)
      throw new InvalidParameterException("Added metal cannot be null");
     myAddedMetal = addedMetal;
  }
}
```

```
/*
 * @(#)Coordinate.java
 *
 * History:  1999-2000: Written by M. Gardner and C. Dalton
 *           Oct 2001:  Rewritten from scratch by G.Lewis.
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

import java.util.*;
import java.io.*;

/** Structural elements (e.g. Columns) are added to the structure.
 */
public class Coordinate implements Serializable
{
  /**The x-position of the coordinate */
  private float myXPosition;
  /**The y-position of the coordinate */
  private float myYPosition;
  /**The z-position of the coordinate */
  private float myZPosition;

  /**Constructor
   */
  public Coordinate(){
  }

  /**Constructor
   * @param xpos  the x-position of the coordinate
   * @param ypos  the y-position of the coordinate
   * @param zpos  the z-position of the coordinate
   */
  public Coordinate(float xpos, float ypos, float zpos)
    throws InvalidParameterException{
    if ((xpos < 0) || (ypos < 0) || (zpos < 0))
      throw new
      InvalidParameterException("A coordinate position was less than zero");
    myXPosition = xpos;
    myYPosition = ypos;
    myZPosition = zpos;
  }

  /**Get the x position of the coordinate */
  public float getXPosition(){
    return myXPosition;
  }

  /**Set the x position of the coordinate
   * @param xpos the position to set the x position to
   */
  public void setXPosition(float xpos) throws InvalidParameterException{
    if (xpos < 0)
      throw new
```

```
        InvalidParameterException("The x coordinate was less than zero");
      myXPosition = xpos;
    }

    /**Get the y position of the coordinate */
    public float getYPosition(){
      return myYPosition;
    }

    /**Set the y position of the coordinate
     * @param ypos the position to set the y position to
     */
    public void setYPosition(float ypos) throws InvalidParameterException{
      if (ypos < 0)
        throw new
        InvalidParameterException("The y coordinate was less than zero");
      myYPosition = ypos;
    }

    /**Get the z position of the coordinate */
    public float getZPosition(){
      return myZPosition;
    }

    /**Set the z position of the coordinate
     * @param zpos the position to set the x position to
     */
    public void setZPosition(float zpos) throws InvalidParameterException{
      if (zpos < 0)
        throw new
        InvalidParameterException("The z coordinate was less than zero");
      myZPosition = zpos;
    }

    /**Parse a string representation of a coordinate */
    public static Coordinate parseCoordinate(String coord)
      throws InvalidParameterException
    {
      StringTokenizer st = new StringTokenizer(coord,"(),");
      if (st.countTokens() != 3)
        throw new InvalidParameterException("Exactly three coordinates
  required");

      float xpos;
      try{
         xpos = Float.parseFloat(st.nextToken());
        if (xpos < 0)
        throw new InvalidParameterException("x position must be greater than or
  equal to 0.");
      }
      catch (NumberFormatException e){
        throw new InvalidParameterException("x position invalid.");
      }

      float ypos;
      try{
         ypos = Float.parseFloat(st.nextToken());
```

```
      if (ypos < 0)
      throw new InvalidParameterException("y position must be greater than or
equal to 0.");
    }
    catch (NumberFormatException e){
      throw new InvalidParameterException("y position invalid.");
    }

    float zpos;
    try{
       zpos = Float.parseFloat(st.nextToken());
      if (zpos < 0)
      throw new InvalidParameterException("z position must be greater than or
equal to 0.");
    }
    catch (NumberFormatException e){
      throw new InvalidParameterException("z position invalid.");
    }

    return new Coordinate(xpos,ypos,zpos);

  }

  /**Is this corrdinate equal to a given coordinate */
  public boolean isEqual(Coordinate c){
    return ((myXPosition == c.getXPosition()) &&
           (myYPosition == c.getYPosition()) &&
           (myZPosition == c.getZPosition()));
  }

  /**get a string representation of the object */
  public String toString(){
    return "(" + getXPosition() + "," + getYPosition() + "," +
      getZPosition() +")";
  }

}
```

```
/*
 * @(#)dblComparator.java
 *
 * History:  Mar 2002: Written by M. Gardner and G. Lewis
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;
import java.util.*;
import java.io.*;

/**Class for the set comparator */
class dblComparator implements Comparator, Serializable{
    /**Compares its two arguments for order*/
    public int compare(Object o1,Object o2){
      if(!(o1 instanceof Double))
          throw new ClassCastException();
      if(!(o2 instanceof Double))
          throw new ClassCastException();
      double dbl = ((Double)o1).doubleValue() - ((Double)o2).doubleValue();
      return (int) dbl;
    }
}
```

```
/*
 * @(#) EditElementDialog.java
 *
 * History:  Nov 2001: Written by M. Gardner and G. Lewis
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

import javax.swing.*;
import javax.swing.border.*;
import javax.accessibility.*;

import java.awt.*;
import java.awt.event.*;

import structuralAnalyser.structuralElement.*;
import structuralAnalyser.material.*;

public class EditElementDialog extends JDialog {
  /** The element being edited*/
  StructuralElement structuralElement = null;
  /**The day the element was constructed*/
  int dayConstructed = -1;
  /**The starting coorindate of the element */
  Coordinate startCoordinate = null;

  /**The parent frame */
  AnalyserFrame myParent = null;

  /** The different elements that can be constructed */
  private String[] elementStrings = { " Column "};
    //" Beam "," Floor ", " Wall " // for now only allow editing of columns

  /** Panel for entering column details */
  ColumnPanel myColumnPanel = null;
  /** Panel for entering beam details */
  BeamPanel myBeamPanel = null;
  /** Panel for entering floor details */
  FloorPanel myFloorPanel = null;
  /** Panel for entering wall details */
  WallPanel myWallPanel = null;
  /** The content pane for this dialog */
  Container contentPane = null;
  /** The panel that is currently being displayed */
  StructuralElementPanel currentPanel = null;
  /** The panel that holds the element panel*/
  JPanel myElementContainerPanel = null;
  /** The pane that displays the cancel and finish buttons*/
  JPanel myCancelFinishPane = null;

  /** Finish Action */
  Action myFinishAction = new FinishAction();
  /** Cancel Action */
  Action myCancelAction = new CancelAction();
```

```
  /**A border that puts 10 extra pixels around a pane.*/
  Border myPaneEdgeBdr = BorderFactory.createEmptyBorder(10,10,10,10);

  /** Constructor
   * @param parent the parent frame */
  public EditElementDialog(AnalyserFrame parent, StructuralElement elt)    {
    super(parent,"Structural Element Editor",true);
    contentPane = getContentPane();
    myParent = parent;

    structuralElement = elt;

    //Set up the column panel
    myColumnPanel  = new ColumnPanel();
    myColumnPanel.setPreferredSize(new Dimension(300, 310));
    myColumnPanel.setOpaque(true);
    myColumnPanel.setForeground(Color.black);

    //Set up the beam panel
    myBeamPanel  = new BeamPanel();
    myBeamPanel.setPreferredSize(new Dimension(300, 310));
    myBeamPanel.setOpaque(true);
    myBeamPanel.setForeground(Color.black);

    //Set up the floor panel
    myFloorPanel  = new FloorPanel();
    myFloorPanel.setPreferredSize(new Dimension(300, 310));
    myFloorPanel.setOpaque(true);
    myFloorPanel.setForeground(Color.black);

    //Set up the wall panel
    myWallPanel  = new WallPanel();
    myWallPanel.setPreferredSize(new Dimension(300, 310));
    myWallPanel.setOpaque(true);
    myWallPanel.setForeground(Color.black);

    //Add control pane and panel to frame.
    contentPane.setLayout(new BoxLayout(contentPane,
                                        BoxLayout.Y_AXIS));
    contentPane.add(Box.createRigidArea(new Dimension(0, 10)));
    contentPane.add(createControlPanel());
    contentPane.add(Box.createRigidArea(new Dimension(0, 10)));
    myElementContainerPanel = new JPanel();

myElementContainerPanel.setBorder(BorderFactory.createLineBorder(Color.black))
;
    myElementContainerPanel.add(myColumnPanel);
    contentPane.add(myElementContainerPanel);
    currentPanel = myColumnPanel;
    //add the cancel and finish buttons
    myCancelFinishPane = new CancelFinishPane();
    contentPane.add(Box.createRigidArea(new Dimension(0, 10)));
    contentPane.add(myCancelFinishPane);
  }

  /**Abstract panel for entering structural elements details */
```

```
  abstract class StructuralElementPanel extends JPanel{
    /**The day the element was constructed */
    private JLabel dayLabel = null;
    private JTextField dayField = null;
    /**The material of the element */
    protected Material material = null;
    private JLabel materialLabel = null;
    private JComboBox materialBox = null;
    /**The starting position of the element */
    private JLabel startCoordLabel = null;
    private JTextField startCoordField = null;

    /**Constructor
     */
    public StructuralElementPanel(){
        setLayout(new GridLayout(0,1));
        JPanel dayPanel = new JPanel();
      dayLabel = new JLabel("Day");
      dayField = new JTextField(4);

dayField.setText(Integer.toString(structuralElement.getDayConstructed()));
      dayField.setEditable(false);
      dayPanel.add(dayLabel);
      dayPanel.add(dayField);

      JPanel startCoordPanel = new JPanel();
      startCoordLabel = new JLabel("Starting Coordinate (x,y,z)");
      startCoordField = new JTextField(10);

startCoordField.setText(structuralElement.getStartCoordinate().toString());
      startCoordField.setEditable(false);
      startCoordPanel.add(startCoordLabel);
      startCoordPanel.add(startCoordField);

      JPanel materialPanel = new JPanel();
      materialLabel = new JLabel("Material");
      materialBox = new JComboBox(Structure.getInstance().getMaterials());
      materialBox.setSelectedItem(structuralElement.getMaterial());
      materialPanel.add(materialLabel);
      materialPanel.add(materialBox);

      add(dayPanel);
      add(materialPanel);
      add(startCoordPanel);
    }

    public boolean parseInput(){

      material = (Material) materialBox.getSelectedItem();
      if (material == null){
        JOptionPane.showMessageDialog(this,
                                      "You must specify a material.",
                                      "Error",
                                      JOptionPane.ERROR_MESSAGE);
        return false;
      }
```

```
    try{
        structuralElement.setMaterial(material);
    }
    catch(InvalidParameterException e){
        JOptionPane.showMessageDialog(this,
                                      "Invalid material",
                                      "Error",
                                      JOptionPane.ERROR_MESSAGE);
    }
    return true;
  }
}


/**Panel for entering in a columns details */
class ColumnPanel extends StructuralElementPanel{
  /**The length of the column */
  private JLabel lengthLabel = null;
  private JTextField lengthField = null;
  /**The section of the column */
  private Section section = null;
  private JLabel sectionLabel = null;
  private JComboBox sectionBox = null;

  /**Constructor */
  public ColumnPanel()
  {
    super();
    JPanel lengthPanel = new JPanel();
    lengthLabel = new JLabel("Length");
    lengthField = new JTextField(4);
    Column col = (Column) structuralElement;
    lengthField.setText(Float.toString(col.getLength()));
    lengthField.setEditable(false);
    lengthPanel.add(lengthLabel);
    lengthPanel.add(lengthField);

    JPanel sectionPanel = new JPanel();
    sectionLabel = new JLabel("Section");
    sectionBox = new JComboBox(Structure.getInstance().getSections());
    sectionBox.setSelectedItem(col.getSection());
    sectionPanel.add(sectionLabel);
    sectionPanel.add(sectionBox);

    add(lengthPanel);
    add(sectionPanel);
  }

  /**Parse the input typed by the user. Returns true if the input
   *   was successfully parsed.
   */
  public boolean parseInput(){
    if (!super.parseInput())
      return false;

    section = (Section) sectionBox.getSelectedItem();
    if (section == null){
```

```
        JOptionPane.showMessageDialog(this,
                                      "You must specify a section for the
column",
                                      "Error",
                                      JOptionPane.ERROR_MESSAGE);
        return false;
      }

      try{
          Column col = (Column) structuralElement;
          col.setSection(section);
      }
      catch(InvalidParameterException e){
          JOptionPane.showMessageDialog(this,
                                        "Invalid section",
                                        "Error",
                                        JOptionPane.ERROR_MESSAGE);
      }

      return true;
    }
  }

  /**Panel for entering in beam details */
  class BeamPanel extends StructuralElementPanel{
    /**The section of the beam */
    private Section section = null;
    private JLabel sectionLabel = null;
    private JComboBox sectionBox = null;

    /**The ending position of the beam */
    private Coordinate endCoordinate  = null;
    private JLabel endCoordLabel = null;
    private JTextField endCoordField = null;

    /**The fixed position of the beam */
    private Coordinate fixedCoordinate  = null;
    private JLabel fixedCoordLabel = null;
    private JTextField fixedCoordField = null;

    public BeamPanel()
    {
        super();
      JPanel sectionPanel = new JPanel();
      sectionLabel = new JLabel("Section");
      sectionBox = new JComboBox(Structure.getInstance().getSections());
      sectionPanel.add(sectionLabel);
      sectionPanel.add(sectionBox);

      JPanel endCoordPanel = new JPanel();
      endCoordLabel = new JLabel("Ending Coordinate (x,y,z)");
      endCoordField = new JTextField(10);
      endCoordPanel.add(endCoordLabel);
      endCoordPanel.add(endCoordField);

      JPanel fixedCoordPanel = new JPanel();
      fixedCoordLabel = new JLabel("Fixed Coordinate (x,y,z)");
```

```
    fixedCoordField = new JTextField(10);
    fixedCoordPanel.add(fixedCoordLabel);
    fixedCoordPanel.add(fixedCoordField);

    add(sectionPanel);
    add(endCoordPanel);
    add(fixedCoordPanel);
  }

  /**Parse the input typed by the user. Returns true if the input
   *   was successfully parsed.
   */
  public boolean parseInput(){
      return false; // not implemented yet
  }
}

/**Panel for entering in floor details */
class FloorPanel extends StructuralElementPanel{
  /**The thickness of the floor*/
  private JLabel thicknessLabel = null;
  private JTextField thicknessField = null;

  /**The ending position of the floor */
  private Coordinate endCoordinate  = null;
  private JLabel endCoordLabel = null;
  private JTextField endCoordField = null;

  public FloorPanel()
  {
      super();
    JPanel thicknessPanel = new JPanel();
    thicknessLabel = new JLabel("Thickness");
    thicknessField = new JTextField(4);
    thicknessPanel.add(thicknessLabel);
    thicknessPanel.add(thicknessField);

    JPanel endCoordPanel = new JPanel();
    endCoordLabel = new JLabel("Ending Coordinate (x,y,z)");
    endCoordField = new JTextField(10);
    endCoordPanel.add(endCoordLabel);
    endCoordPanel.add(endCoordField);

    add(thicknessPanel);
    add(endCoordPanel);
  }

  /**Parse the input typed by the user. Returns true if the input
   * was successfully parsed.
   */
  public boolean parseInput(){
      return false; //not implemented yet
  }
}

/**Panel for entering in wall details */
class WallPanel extends StructuralElementPanel{
```

```
    public WallPanel()
    {
        super();
      JLabel lbl = new JLabel("Wall specific details to go here.");
      add(lbl);
    }
  }

  //Create the control pane for the top of the frame.
  private JPanel createControlPanel() {
    final JComboBox layerList = new JComboBox(elementStrings);
    layerList.setSelectedIndex(0);
    layerList.addActionListener(new ActionListener () {
        public void actionPerformed(ActionEvent e) {
          int index = layerList.getSelectedIndex();
          JPanel oldPanel = currentPanel;
          if (index == 0)
            currentPanel = myColumnPanel;
          else if (index == 1)
            currentPanel = myBeamPanel;
          else if (index == 2)
            currentPanel = myFloorPanel;
          else if (index == 3)
            currentPanel = myWallPanel;
          if (oldPanel != currentPanel){
            oldPanel.setVisible(false);
            myElementContainerPanel.add(currentPanel);
            currentPanel.setVisible(true);
          }
        }
      });

    JPanel controls = new JPanel();
    controls.add(layerList);
    controls.setBorder(BorderFactory.createTitledBorder("Choose the structural
element."));
    return controls;
  }


  /**
   * The cancel/finish pane. Displays the cancel and finish
   * buttons.
   */
  private class CancelFinishPane extends JPanel{
    /** Finish Button */
    JButton myFinishBtn = null;
    /** Cancel Button */
    JButton myCancelBtn = null;

    /**Constructor*/
    public CancelFinishPane(){
      // create the buttons and register the actions
      myCancelBtn = new JButton("Cancel");
      myCancelBtn.addActionListener(myCancelAction);
      myCancelBtn.setToolTipText("Click to cancel.");
      myFinishBtn = new JButton("Finish");
```

```
        myFinishBtn.addActionListener(myFinishAction);
        myFinishBtn.setToolTipText("Click to finish.");

        //add the buttons to the pane
        BoxLayout layout = new BoxLayout(this, BoxLayout.X_AXIS);
        setLayout(layout);
        setBorder(myPaneEdgeBdr);
        add(Box.createHorizontalGlue());
        add(myCancelBtn);
        add(Box.createRigidArea(new Dimension(10, 0)));
        add(myFinishBtn);
      }
    }

    /**
     * Inner class that defines the cancel action
     */
    private class CancelAction extends AbstractAction {
      public void actionPerformed(ActionEvent evt) {
        setVisible(false);
      }
    }


    /**
     * Inner class that defines the finish action
     */
    private class FinishAction extends AbstractAction {
      public void actionPerformed(ActionEvent evt) {
        if (currentPanel.parseInput()){
            setVisible(false);
        }
      }
    }
  }
```

```
/*
 * @(#)Framing.java
 *
 * History:  Apr 2002    Written by M. Gardner and G. Lewis
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;
import java.io.*;
import java.util.*;
import structuralAnalyser.structuralElement.*;

/** A Column structural element */
public class Framing implements Serializable {

    /** The floor number this framing is applied to (the framing is
     * applied to the columns below this floor) */
    protected int myFloorNumber = -1;
    /**The day the framing was applied to the structure*/
    protected int myDayApplied = -1;
    /**The framing data, stored as a 2D array */
    protected double[][] myFramingData;

    /**Construct a new framing
     * @param floorNumber the floor the framing was to be applied to
     * (framing is applied below this floor)
     * @param day the day the framing was added
     * @param framing the framing table as a vector of arrays (each
     * array being a column of the table)
     */
  public Framing(int floorNumber, int numColumnsOnFloor)
  {
    setFloorNumber(floorNumber);
    myFramingData= new double[numColumnsOnFloor][numColumnsOnFloor];
  }

  /**Get the day the floor number */
  public int getFloorNumber()
  {
    return myFloorNumber;
  }

    /**Set the floor number */
  public void setFloorNumber(int floorNumber)
  {
    myFloorNumber = floorNumber;
  }

  /** Get the day the framing was applied to the structure */
  public int getDayApplied()
  {
    return myDayApplied;
  }

  /**Set the day the framing was applied to the structure
```

```
   * @param dayApplied the day the framing was applied to the structure
   */
  public void setDayApplied(int dayApplied)
      throws InvalidParameterException{
      if (dayApplied < 0)
        throw new InvalidParameterException("Attempting to add framing on an
invalid day.");
    myDayApplied = dayApplied;
  }

  /** Set the framing value
   * @param row the row for the value
   * @param col the column for the value
   * @param value the framing value
   */
  public void setValue(int row, int col, double value)
  {
            myFramingData[row][col] = value;
  }

  /** Get the framing vector
   * @param row the row for the value
   * @param col the column for the value
   */
  public double getValue(int row, int col)
  {
            return myFramingData[row][col];
  }
    /**Check if the framing data is valid. Data will be invalid if
     * columns have been addded to the floor since framing was
     * entered, or if some framing values are invalid or have not been
     * entered.*/
    public boolean isDataValid(){
      if (myFloorNumber < 1)
          return false;
      int numColumns =
          Structure.getInstance().getNumColumns(new Integer(myFloorNumber-1));
      if(numColumns != myFramingData.length)
          return false;
      return isDataComplete();
    }
    /**Check if the framing data is complete */
    public boolean isDataComplete(){
      return true; // for now
    }
    /** Get the number of columns of framing data */
    public int getNumberColumns(){
      return myFramingData.length;
    }
    /** Get the number of rowans of framing data */
    public int getNumberRows(){
      return myFramingData[0].length;
    }

}
```

```
/*
 * @(#)FramingAlgorithm.java
 *
 * History:   Mar 2002: Written by M. Gardner and G. Lewis
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

import structuralAnalyser.structuralElement.*;
import structuralAnalyser.material.*;

import java.util.*;
import java.io.*;
import java.lang.Math.*;
import Jama.*;
import java.*;

/**
 * This class contains the algorithm for calculating the shrinkage of
 * columns when framing is taken into consideration. This class is
 * a singleton.
 */
public class FramingAlgorithm extends ShrinkageAlgorithm{

    /**The instance of this class */
    private static FramingAlgorithm myInstance = null;

    /** The tolerance for comparing deltas */
    double myTolerance = 0.85;
  int counter = 0;
    public static ShrinkageAlgorithm getInstance(){
      if (myInstance ==  null)
          myInstance = new FramingAlgorithm();
      return myInstance;
    }


  // ALL calcColumndelta METHODS  NEED TO RETURN A VECTOR OF COLUMNS WITH
tempDelta SET.

    public Vector calcColumnDelta(Vector columns, int day)
      throws InvalidParameterException{

      /** A set of integers representing the (heights of) floors
       * that need shrinkages calculated. */
        SortedSet floors = new TreeSet(new dblComparator());


      // 1  SORT THE COLUMNS INTO THE CORRECT ORDER FOR THE MATRIX

      Vector sortedColumns = new Vector();
      Vector sortedStacks = Structure.getInstance().getSortedStack(columns);
      System.out.println("sortedStacks at start of calcColumnDelta
FramingAlgorithm " + sortedStacks);
```

```
        Vector firstStack = (Vector)sortedStacks.elementAt(0);
        //System.out.println("firstStack " + firstStack);

        for (int s = 0; s < firstStack.size(); s++){

          for (int t = 0; t < sortedStacks.size(); t++){
            Vector aStack = (Vector)sortedStacks.elementAt(t);
            sortedColumns.addElement((Column)aStack.elementAt(s));
          }
        }
        //System.out.println("sortedColumns " + sortedColumns);


        //find the (heights of) floors that need shrinkages calculated
        for (int i = 0; i < columns.size(); i++){
            Column col = (Column) columns.elementAt(i);
            col.setTempDelta(0);
            col.setTempDeltaExist(0);
            col.setTempDeltaNew(0);
            col.setTempCreep(0);
            col.setTempCreepPrevious(0);
            float length = col.getLength();
            Coordinate startCoord = col.getStartCoordinate();
            floors.add(new Double(startCoord.getYPosition()));
        }

        //Get the columns on each floor. Stored as a vector of
        //vectors. Each vector is the columns for a given floor
        Vector floorColumnsVector = new Vector();
        int numFloors = Structure.getInstance().getNumFloors();
        for (int i =0; i < numFloors-1; i++)
            floorColumnsVector.add(Structure.getInstance().getColumns(new
Integer(i)));

        int numColumnsOnFloor =
            ((Vector) floorColumnsVector.elementAt(0)).size();

        //check whether the framing data has been entered for each floor
        boolean framingDataEntered = true;
        for (int i =1; i < numFloors; i++){
            Framing framing = Structure.getInstance().getFraming(new
Integer(i));
            if (!framing.isDataValid()){
              framingDataEntered = false;
              break;
            }
        }
        //assert that the framing data has been entered. Should handle
        //this gracefully at some stage
        Assert.assert(framingDataEntered, "Framing data is not complete");

        //Now implement the algorithm
        //first find all stages.
        SortedSet stages = Structure.getInstance().getStages();
        //add the day of the calculation, and remove any stages past that day
        Integer theDay = new Integer(day);
        Integer theDayAfter = new Integer(day+1);
```

```
stages.add(theDay);
stages = stages.headSet(theDayAfter);
//stages.add(theDay);

//find the shrinkage for each stage.
Matrix A = buildA(floorColumnsVector, numColumnsOnFloor);
//System.out.println("A = "); A.print(4,2);
Object[] stagesArray = stages.toArray();
//System.out.println("Stages "+stages);


// DONT NEED THIS ???
//the total shrinkages found so far
//Vector totalDeltas = new Vector();
//initialise totals to 0
//for (int i = 0; i < numColumnsOnFloor*floorColumnsVector.size(); i++)
// totalDeltas.addElement(new Double(0));




//the old deltas for existing loads of a given stage
Vector oldDeltasExist = new Vector();
//the old deltas for new loads of a given stage
Vector oldDeltasNew = new Vector();
//the new deltas for  existing loads of a given stage
Vector newDeltasExist = null;
//the new deltas for new loads of a given stage
Vector newDeltasNew = null;


for (int i = 0; i < (stagesArray.length-1); i++){
    int startDay = ((Integer) stagesArray[i]).intValue();
    int dayBeforeEnd = ((Integer) stagesArray[i+1]).intValue() - 1;
    int endDay = ((Integer) stagesArray[i+1]).intValue();
    //get the columns that were constructed by the startDay
    Vector constructedColumns = new Vector();
    //Vector columnsNotConstructed = new Vector();
    System.out.println("Stage no " +i);
    System.out.println("StartDay " +startDay);
    //System.out.println("day before end " +dayBeforeEnd);
    System.out.println("endDay " +endDay);
    //System.out.println(columns);
    oldDeltasExist.removeAllElements();
    oldDeltasNew.removeAllElements();


    for (int j = 0; j < sortedColumns.size(); j++){
      Column c = (Column)sortedColumns.elementAt(j);
      c.setTempDeltaExist(0);
      c.setTempDeltaNew(0);

      c.setConstructed(false);
      oldDeltasExist.addElement(new Double(0));
      oldDeltasNew.addElement(new Double(0)) ;
```

```
            if (c.getDayConstructed() <= startDay)
                c.setConstructed(true);
            constructedColumns.addElement(c);
            }

          //System.out.println(constructedColumns + "constructedColumns to be
used in Matrix equations");
            //System.out.println(oldDeltasExist);

          Matrix PEE =  buildPEExist(floorColumnsVector,numColumnsOnFloor, 0,
                        dayBeforeEnd);
          //          System.out.println("PEE = "); PEE.print(4,2);
          Matrix PEN = buildPENew(floorColumnsVector,numColumnsOnFloor,
dayBeforeEnd,
                        endDay);
          // System.out.println("PEN = "); PEN.print(4,2);
          Matrix KE = buildKExist(floorColumnsVector,
numColumnsOnFloor,dayBeforeEnd );
          // System.out.println("KE = "); KE.print(4,2);
          Matrix KN = buildKNew(floorColumnsVector, numColumnsOnFloor,
endDay);
          // System.out.println("KN = "); KN.print(4,2);

          newDeltasExist = null;
          newDeltasNew = null;

 System.out.println("********************* started loop to calc deltas
exists************* ");
 while ((newDeltasExist == null) ||
!deltasEqual(oldDeltasExist,newDeltasExist)){
    //while ((newDeltasExist == null) || ( counter < 50)){
            //average old and new deltas
            if (newDeltasExist != null)
                oldDeltasExist = averageDeltas(oldDeltasExist,newDeltasExist);

            if (counter < 30 ){
              counter++;
                  //System.out.println(counter + " counter exist");
              Matrix D = buildD(floorColumnsVector, numColumnsOnFloor,
            oldDeltasExist);

            //System.out.println("D = "); D.print(4,4);
            Matrix KD = KE.times(D);
            //System.out.println("KE x D = "); KD.print(4,4);
            Matrix PEKD = PEE.plus(KD);
            System.out.println("PEE + KD = "); PEKD.print(4,4);
            //Matrix PI = PEKD;
            Matrix PI = A.solve(PEKD);
            System.out.println("PI = "); PI.print(4,4);
            //add internal loads to columns
            Vector internalLoads = getInternalLoads(floorColumnsVector,
                                        numColumnsOnFloor,PI,
                                        startDay);
            //calculate new deltas. A vector of columns will be
            //returned. The delta is an attribute on the
column(tempDeltaExist)
            newDeltasExist = new Vector();
```

```
            Vector newDeltasValues =
NonFramingAlgorithm.getInstance().calcColumnDeltaExist(constructedColumns,star
tDay,endDay,internalLoads);

for (int m = 0; m<newDeltasValues.size(); m++){
              double d =
((Column)newDeltasValues.elementAt(m)).getTempDeltaExist();
               newDeltasExist.addElement(new Double(d));
            }
//System.out.println("newDeltasExists as calculated by calcColumnDeltaExist
"+newDeltasExist);


            if (deltasEqual(oldDeltasExist,newDeltasExist) || (counter == 30
)){
              System.out.println("inSetTempCreepPrevious, last loop ");
              for (int l = 0; l < constructedColumns.size(); l++){
                Column tempC = (Column)constructedColumns.elementAt(l);
                //System.out.println("TempCreepPrevious " +
tempC.getTempCreepPrevious());
                tempC.setTempCreepPrevious(tempC.getTempCreep());
                //System.out.println("TempCreepPrevious " +
tempC.getTempCreepPrevious());
                //System.out.println("TempCreep " + tempC.getTempCreep());
              }
            }
            }
            else{
              System.out.println(" should end loop exist");
              oldDeltasExist = newDeltasExist ;
            }

          }// End While
          counter = 0;

 System.out.println("********************* started loop to calc deltas
New************* ");

 while ((newDeltasNew == null) || !deltasEqual(oldDeltasNew,newDeltasNew)){
   //while ((newDeltasNew == null) || ( counter < 50)){
            //average old and new deltas
            if (newDeltasNew != null)
                oldDeltasNew = averageDeltas(oldDeltasNew,newDeltasNew);

            counter++;

            if (counter < 30 ){
            //System.out.println(counter + " counter new");
            Matrix D = buildD(floorColumnsVector, numColumnsOnFloor,
                        oldDeltasNew);
            //System.out.println("D = "); D.print(4,4);
            Matrix KD = KN.times(D);
            //System.out.println("KN x D = "); KD.print(4,4);
            Matrix PEKD = PEN.plus(KD);
            System.out.println("PEN + KD = "); PEKD.print(4,4);
            //Matrix PI = PEKD;
            Matrix PI = A.solve(PEKD);
```

```
            System.out.println("PI = "); PI.print(4,4);
            //add internal loads to columns
            Vector internalLoads =
getInternalLoads(floorColumnsVector,numColumnsOnFloor,PI, startDay);
            //calculate new deltas. A vector of columns will be
            //returned. The delta is an attribute on the column(tempDeltaNew)
            newDeltasNew = new Vector();
            Vector newDeltasValues =
NonFramingAlgorithm.getInstance().calcColumnDeltaNew(constructedColumns,startD
ay,endDay,internalLoads);
            for (int m = 0; m<newDeltasValues.size(); m++){
              double d =
((Column)newDeltasValues.elementAt(m)).getTempDeltaNew();
               newDeltasNew.addElement(new Double(d));
               }
            //System.out.println("newDeltasNew as calculated by
calcColumnDeltaNew  "+newDeltasNew);
            }

            else{
              System.out.println(" should end loop new");
              oldDeltasNew = newDeltasNew;
            }
          }// End While
          counter = 0;
          // At this stage we have just calculated the new and existing deltas
for a stage in the total age of the structure. The actual total delta for the
stage is the sum of the two values


          // set all the columns tempDelta to be sum of tempExist and tempNew
OR if not constructed set equal to the column below.
          //System.out.println("Add together all the tempDeltas or set the
value for not constructed columns");
          for(int k = 0; k< constructedColumns.size(); k++){
            Column aColumn = (Column)constructedColumns.elementAt(k);
            //System.out.println("a column from the constructedColumns  " +
aColumn);
            double currentTemp = 0;
            if (aColumn.getConstructed()){
            currentTemp = aColumn.getTempDelta();
            aColumn.setTempDelta(currentTemp +  aColumn.getTempDeltaExist() +
aColumn.getTempDeltaNew());
            //System.out.println("total delta to be set = curretTemp +
deltaExist + DeltaNew  " + currentTemp +" "+ aColumn.getTempDeltaExist() +" "+
aColumn.getTempDeltaNew() + "  =  " +  aColumn.getTempDelta());
            }

            else if(Structure.getInstance().getColumnUnder(aColumn) != null){

             Column colUnder =
Structure.getInstance().getColumnUnder(aColumn);
             aColumn.setTempDelta(colUnder.getTempDelta());
             //System.out.println("Column Under not constructed and tempDelta
" + colUnder + colUnder.getTempDelta());
             //System.out.println("not constructed column temp Delta  " +
aColumn.getTempDelta());
```

```
            }
          }
          counter = 0;
          System.out.println("*********End of Stage*********");


      }// End For (i as variable) All stages have been calculated.
      System.out.println("*********End of all Stages*********");
      counter = 0;
        //System.out.println("Columns to be returned  " + columns);
      return columns;
    }// end of method



    /**Build the matix A, used to calculate the deltas
     * @param floorColumnsVector the columns on each floor (a vector
     * of vectors)
     * @param numColumnsOnFloor the number of columns on each floor
     */
    public Matrix buildA(Vector floorColumnsVector, int numColumnsOnFloor){
      //Find the total size of A
      Assert.assert(floorColumnsVector.size() > 0);
      int size = floorColumnsVector.size() * numColumnsOnFloor;
      Matrix A = new Matrix(size,size);

      for (int m=0; m < size; m++){
        for (int n=0; n < size; n++){
          if (n == m)
            A.set(m,n,1);
          else if (n == (m + numColumnsOnFloor))
            A.set(m,n,-1);
        }
      }

      return A;
    }

    /** build the matrix PEE
     * @param floorColumnsVector the columns on each floor (a vector
     * of vectors)
     * @param numColumnsOnFloor the number of columns on each floor
     * @param startDay the start of this stage
     * @param endDay the end of this stage
     */
    public Matrix buildPEExist(Vector floorColumnsVector, int
numColumnsOnFloor,
                    int startDay, int endDay){
      Matrix PEE= new Matrix(numColumnsOnFloor*floorColumnsVector.size(),1);
      //assume that the columns are in the correct order on each floor.
      for (int i =0; i < floorColumnsVector.size(); i++){
          //get the columns on the floor
          Vector floorColumns = (Vector) floorColumnsVector.elementAt(i);
          for (int j = 0; j < floorColumns.size(); j++){
            Column c = (Column) floorColumns.elementAt(j);
```

```
            double totalLoad = 0;
            if (c.getDayConstructed() <= endDay){// changed here
              //System.out.println( "this is one of the columns in PEE up to
endDay for stage " +c);
              Vector columnLoads =
Structure.getInstance().getAllLoads(0,endDay,c);
              //System.out.println( "these are the loads up to endDay for
stage for this column " +c);
                if (columnLoads != null){
                  for (int k =0; k < columnLoads.size(); k++){
                      Load l = (Load) columnLoads.elementAt(k);
                      totalLoad += l.getWeight();
                  }
                }
            }
            //else
            //  Assert.assert(c.getDayConstructed() >= endDay);
            PEE.set((i*numColumnsOnFloor)+j,0,totalLoad);
          }
      }
      return PEE;
    }

 /** build the matrix PEN
     * @param floorColumnsVector the columns on each floor (a vector
     * of vectors)
     * @param numColumnsOnFloor the number of columns on each floor
     * @param startDay the start of this stage
     * @param endDay the end of this stage
     */
    public Matrix buildPENew(Vector floorColumnsVector, int numColumnsOnFloor,
                    int startDay, int endDay){
      Matrix PEN = new Matrix(numColumnsOnFloor*floorColumnsVector.size(),1);
      //assume that the columns are in the correct order on each floor.
      for (int i =0; i < floorColumnsVector.size(); i++){
          //get the columns on the floor
          Vector floorColumns = (Vector) floorColumnsVector.elementAt(i);
          for (int j = 0; j < floorColumns.size(); j++){
            Column c = (Column) floorColumns.elementAt(j);
            double totalLoad = 0;
            if (c.getDayConstructed() <= startDay){

                Vector columnLoads =
      Structure.getInstance().getAllLoads(startDay,endDay+1,c);
                //System.out.println(columnLoads);
                if (columnLoads != null){
                  for (int k =0; k < columnLoads.size(); k++){
                      Load l = (Load) columnLoads.elementAt(k);
                      totalLoad += l.getWeight();
                  }
                }
            }
            //else
            //Assert.assert(c.getDayConstructed() >= endDay);
            PEN.set((i*numColumnsOnFloor)+j,0,totalLoad);
          }
      }
```

```
      return PEN;
    }

    /** build the matrix KE
     * @param floorColumnsVector the columns on each floor (a vector
     * of vectors)
     * @param numColumnsOnFloor the number of columns on each floor
     */
    public Matrix buildKExist(Vector floorColumnsVector, int
numColumnsOnFloor,
                    int startDay){
      Matrix KE = new Matrix(numColumnsOnFloor*floorColumnsVector.size(),
                        numColumnsOnFloor*floorColumnsVector.size());

      //add k for each floor
      int n = 0;
      int m = 0;
      for (int floor = 1; floor <= floorColumnsVector.size(); floor++){
          Framing framing = Structure.getInstance().getFraming(new
Integer(floor));
          if (framing.getDayApplied() <= startDay){
            Matrix k = new Matrix(numColumnsOnFloor,numColumnsOnFloor);
            for (int km =0; km < numColumnsOnFloor; km++){
                for (int kn =0; kn < numColumnsOnFloor; kn++){
                  if (kn == km){
                      //sum the stiffness values
                      double sum = 0;
                      for (int i = 0; i < numColumnsOnFloor; i++){
                        if (i != kn)
                            sum += framing.getValue(km,i);
                      }
                      k.set(km,kn,-1*sum);
                  }
                  else
                      k.set(km,kn,framing.getValue(km,kn));

                }
            }
            KE.setMatrix(m,(m+numColumnsOnFloor-1),n,(n+numColumnsOnFloor-
1),k);
            n+=numColumnsOnFloor;
            m+=numColumnsOnFloor;
          }
      }
      return KE;
    }


 /** build the matrix KN
     * @param floorColumnsVector the columns on each floor (a vector
     * of vectors)
     * @param numColumnsOnFloor the number of columns on each floor
     */
    public Matrix buildKNew(Vector floorColumnsVector, int numColumnsOnFloor,
                    int startDay){
      Matrix KN = new Matrix(numColumnsOnFloor*floorColumnsVector.size(),
                        numColumnsOnFloor*floorColumnsVector.size());
```

```
        //add k for each floor
        int n = 0;
        int m = 0;
        for (int floor = 1; floor <= floorColumnsVector.size(); floor++){
            Framing framing = Structure.getInstance().getFraming(new
Integer(floor));
            if (framing.getDayApplied() <= startDay){
              Matrix k = new Matrix(numColumnsOnFloor,numColumnsOnFloor);
              for (int km =0; km < numColumnsOnFloor; km++){
                  for (int kn =0; kn < numColumnsOnFloor; kn++){
                    if (kn == km){
                        //sum the stiffness values
                        double sum = 0;
                        for (int i = 0; i < numColumnsOnFloor; i++){
                          if (i != kn)
                              sum += framing.getValue(km,i);
                        }
                        k.set(km,kn,-1*sum);
                    }
                    else
                        k.set(km,kn,framing.getValue(km,kn));

                  }
              }
              KN.setMatrix(m,(m+numColumnsOnFloor-1),n,(n+numColumnsOnFloor-
1),k);
              n+=numColumnsOnFloor;
              m+=numColumnsOnFloor;
            }
        }
        return KN;
    }


    /** Build the Matrix D
     * @param floorColumnsVector the columns on each floor (a vector
     * of vectors)
     * @param numColumnsOnFloor the number of columns on each floor
     * @param oldDeltas a vector of old shrinkages
     */
    public Matrix buildD(Vector floorColumnsVector, int numColumnsOnFloor,
                    Vector oldDeltas){
      Matrix D = new Matrix(numColumnsOnFloor*floorColumnsVector.size(),1);
      for (int i = 0; i < oldDeltas.size(); i++){
        //remove oldDeltas. Use floorColumnsVector. Since this is a vector of
vectors, need another loop here.
        D.set(i,0, ((Double) oldDeltas.elementAt(i)).doubleValue());
      }
      return D;
    }

    /** Add the newly calculated internal loads to the columns
     * @param floorColumnsVector the columns on each floor (a vector
     * of vectors)
     * @param numColumnsOnFloor the number of columns on each floor
     * @param PI the matrix of newly calculated loads
```

```
     */
    public Vector getInternalLoads(Vector floorColumnsVector,
                           int numColumnsOnFloor,
                           Matrix PI, int startDay){
      Vector loads = new Vector();
      try{
          for (int i =0; i < floorColumnsVector.size(); i++){
            //assume that the columns are in the correct order on
            //each floor.
            Vector floorColumns = (Vector) floorColumnsVector.elementAt(i);
            for (int j = 0; j < floorColumns.size(); j++){
                double newLoad = PI.get((i*numColumnsOnFloor)+j,0);
                //if (newLoad > 0){
                  Column col = (Column) floorColumns.elementAt(j);
                  Load ld = new Load(col,startDay,newLoad, false);
                  loads.addElement(ld);
                  // }
            }
          }
      }
      catch(InvalidParameterException e){
          System.err.println(e.getMessage());
          System.exit(-1);
      }
      return loads;
    }
    /** Find the average of the old and the new deltas */
    public Vector averageDeltas(Vector oldDeltas, Vector newDeltas){
      Assert.assert(oldDeltas.size() == newDeltas.size());
      Vector averageDeltas = new Vector();
      for (int i =0; i < oldDeltas.size(); i++){
          double oldDelta = ((Double) oldDeltas.elementAt(i)).doubleValue();
          double newDelta = ((Double) newDeltas.elementAt(i)).doubleValue();
          double averageDelta = (oldDelta + newDelta)/2;
          averageDeltas.addElement(new Double(averageDelta));
      }
      return averageDeltas;
    }
    /**Are the old deltas equal to the new deltas (within tolerance)
     */
    public boolean deltasEqual(Vector oldDeltas, Vector newDeltas){
      Assert.assert(oldDeltas.size() == newDeltas.size());
      for (int i=0; i< oldDeltas.size(); i++){
          double oldDelta = ((Double) oldDeltas.elementAt(i)).doubleValue();
          double newDelta = ((Double) newDeltas.elementAt(i)).doubleValue();
          if (oldDelta/newDelta > 1){
            if (newDelta/oldDelta < myTolerance)
            return false;
          }
          else if (oldDelta/newDelta < myTolerance)
            return false;

            // if (Math.abs(oldDelta-newDelta) > myTolerance)
      }
      return true;
    }
}
```

```
/*
 * @(#)intComparator.java
 *
 * History:  Mar 2002: Written by M. Gardner and G. Lewis
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

import java.util.*;
import java.io.*;

/**Class for the set comparator */
class intComparator implements Comparator, Serializable{
    /**Compares its two arguments for order*/
    public int compare(Object o1,Object o2){
      if(!(o1 instanceof Integer))
          throw new ClassCastException();
      if(!(o2 instanceof Integer))
          throw new ClassCastException();
      return ((Integer)o1).intValue() - ((Integer)o2).intValue();
    }
}
```

```
/*
 * @(#)InvalidLoadException.java
 *
 * History:  Oct 2001: Written by G. Lewis and M. Gardner
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

/** An exception thrown when the load cannot be added to the structure */
public class InvalidLoadException extends Exception{
  public InvalidLoadException(){}
  public InvalidLoadException(String s){super(s);}
}
```

```
/*
 * @(#)InvalidParameterException.java
 *
 * History:  Oct 2001: Written by G. Lewis
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

/** An exception thrown when the type is not known */
public class InvalidParameterException extends Exception{
  public InvalidParameterException(){}
  public InvalidParameterException(String s){super(s);}
}
```

```
/*
 * @(#)InvalidPositionException.java
 *
 * History:  Oct 2001: Written by G. Lewis
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

/** An exception thrown when the type is not known */
public class InvalidPositionException extends Exception{
  public InvalidPositionException(){}
  public InvalidPositionException(String s){super(s);}
}
```

```
/*
 * @(#) JScrollPaneAdjuster.java
 *
 * History:  Oct 2001: Written by G. Lewis
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;
import java.awt.Point;
import javax.swing.JScrollPane;
import javax.swing.JViewport;
import javax.swing.SwingUtilities;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeEvent;

import java.io.Serializable;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.io.IOException;


public class JScrollPaneAdjuster
    implements PropertyChangeListener, Serializable
{
    private JScrollPane pane;

    private transient Adjuster x, y;


    public JScrollPaneAdjuster(JScrollPane pane)
    {
        this.pane = pane;

        this.x = new Adjuster(pane.getViewport(), pane.getColumnHeader(),
Adjuster.X);
        this.y = new Adjuster(pane.getViewport(), pane.getRowHeader(),
Adjuster.Y);

        pane.addPropertyChangeListener(this);
    }


    public void dispose()
    {
        x.dispose();
        y.dispose();

        pane.removePropertyChangeListener(this);
        pane = null;
    }


    public void propertyChange(PropertyChangeEvent e)
```

```
    {
        String name = e.getPropertyName();

        if (name.equals("viewport"))
        {
            x.setViewport((JViewport)e.getNewValue());
            y.setViewport((JViewport)e.getNewValue());
        }
        else if (name.equals("rowHeader"))
        {
            y.setHeader((JViewport)e.getNewValue());
        }
        else if (name.equals("columnHeader"))
        {
            x.setHeader((JViewport)e.getNewValue());
        }
    }


    private void readObject(ObjectInputStream in)
        throws IOException, ClassNotFoundException
    {
        in.defaultReadObject();

        x = new Adjuster(pane.getViewport(), pane.getColumnHeader(),
Adjuster.X);
        y = new Adjuster(pane.getViewport(), pane.getRowHeader(), Adjuster.Y);
    }


    private static class Adjuster
        implements ChangeListener, Runnable
    {
        public static final int X = 1, Y = 2;

        private JViewport viewport, header;
        private int type;


        public Adjuster(JViewport viewport, JViewport header, int type)
        {
            this.viewport = viewport;
            this.header = header;
            this.type = type;

            if (header != null)
                header.addChangeListener(this);
        }

        public void setViewport(JViewport newViewport)
        {
            viewport = newViewport;
        }

        public void setHeader(JViewport newHeader)
        {
            if (header != null)
```

```
                header.removeChangeListener(this);

            header = newHeader;

            if (header != null)
                header.addChangeListener(this);
        }

        public void stateChanged(ChangeEvent e)
        {
            if (viewport == null || header == null)
                return;

            if (type == X)
            {
                if (viewport.getViewPosition().x !=
header.getViewPosition().x)
                    SwingUtilities.invokeLater(this);
            }
            else
            {
                if (viewport.getViewPosition().y !=
header.getViewPosition().y)
                    SwingUtilities.invokeLater(this);
            }
        }

        public void run()
        {
            if (viewport == null || header == null)
                return;


            Point v = viewport.getViewPosition(),
                h = header.getViewPosition();

            if (type == X)
            {
                if (v.x != h.x)
                    viewport.setViewPosition(new Point(h.x, v.y));
            }
            else
            {
                if (v.y != h.y)
                    viewport.setViewPosition(new Point(v.x, h.y));
            }
        }

        public void dispose()
        {
            if (header != null)
                header.removeChangeListener(this);

            viewport = header = null;
        }
    }
}
```

```
/*
 * @(#)Load.java
 *
 * History:  1999-2000: Written by M Gardner and C. Dalton
 *           Oct 2001:  G.Lewis added comments, deleted obsolete code,
 *                and changed variable names.
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;
import java.io.*;
import structuralAnalyser.structuralElement.*;

/** A Column structural element */
public class Load implements Serializable {

  /** The element this load is applied to */
  protected StructuralElement myAppliedElement;
  /**The day the load was applied to the structure*/
  protected int myDayConstructed;
  /**The weight of the structure */
  protected double myWeight;
  /** Is this load a new load, or an existing load*/
    private boolean myIsNew;

  /**Construct a new load
   * @param element the element the load is to be applied to
   * @param day the day the load was added
   * @param weight the weight of the load
   */
  public Load(StructuralElement element, int day,
            double weight, boolean isNew) throws InvalidParameterException
  {
    setAppliedElement(element);
    setDayConstructed(day);
    setWeight(weight);
    setIsNew(isNew);
  }

  /**Get the length of this column */
  public StructuralElement getAppliedElement()
  {
    return myAppliedElement;
  }

  /**Set the length of this column */
  public void setAppliedElement(StructuralElement element)
    throws InvalidParameterException
  {
    if (element == null) // need to check  if element exists
      throw new InvalidParameterException("Attempting to add load to null
element.");
    myAppliedElement = element;
  }
```

```
  /** Get the day this element was added to the structure */
  public int getDayConstructed()
  {
    return myDayConstructed;
  }

  /**Set the day this element was added to the structure
   * @param dayConstructed the day this element was added to the structure
   */
  public void setDayConstructed(int dayConstructed){
    myDayConstructed = dayConstructed;
  }

    /** Get the weight of this structural element */
  public double getWeight()
  {
    return myWeight;
  }

  /** Set the material of this element
   * @param m the material to set it to
   */
  public void setWeight(double w)
  {
            myWeight = w;
  }

    /** Get whether this load is new or exiting */
  public boolean getIsNew()
  {
    return myIsNew;
  }

  /** Set the material of this element
   * @param m the material to set it to
   */
  public void setIsNew(boolean isNew)
  {
            myIsNew = isNew;
  }

  public String getToolTipText(){
    return "Click on this load, then click \"Delete Load\" to remove it.";
  }

  public String toString(){
    return "Day: " + getDayConstructed() + " Weight: " + getWeight();
  }

}
```

```
/*
 * @(#)LoadsDialog.java
 *
 * History:  Nov 2001:  Written by M. Gardner and G. Lewis
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;

import structuralAnalyser.structuralElement.*;
import structuralAnalyser.material.*;

/** The dialog for entering loads. */
public class LoadsDialog extends JDialog {
  /**The structural element this load applies to */
  StructuralElement myStructuralElement = null;
  /**The pane that displays the buttons (new load, delete) */
  ButtonsPane myButtonsPane = null;
  /**The pane that displays the list of elements in the structure */
  LoadsPane myLoadsPane = null;

  //Actions used in all panes
  /** New load action */
  Action myNewLoadAction  = new NewLoadAction();
  /** Delete action */
  Action myDeleteAction  = new DeleteAction();
  /** Done action */
  Action myDoneAction  = new DoneAction();

  //Borders used in all panes.
  /**A border that puts 10 extra pixels around a pane.*/
  Border myPaneEdgeBdr = BorderFactory.createEmptyBorder(10,10,10,10);
  /**Etched border, used for adding border to components.*/
  Border myEtchedBdr = BorderFactory.createEtchedBorder();


  /** Update the state of the buttons */
  Runnable refresh = new Runnable() {
      public void run(){
      myButtonsPane.refresh();
      }
    };

  /**Constructor
     @param parent the owner of this dialog
     @param structuralElement the structuralElement this load applies to.
   */
  public LoadsDialog(JFrame parent, StructuralElement structuralElement) {
```

```
    super(parent, "Loads Editor", true);

    myStructuralElement = structuralElement;
    JPanel content = new JPanel();
    content.setLayout(new BoxLayout(content, BoxLayout.Y_AXIS));
    content.setOpaque(false);

    myLoadsPane = new LoadsPane();
    content.add(myLoadsPane, BorderLayout.CENTER);

    myButtonsPane = new ButtonsPane();
    content.add(myButtonsPane, BorderLayout.SOUTH);

    getContentPane().setLayout(new FlowLayout());
    getContentPane().add(content);

    //add in exsiting loads
    Vector loads = Structure.getInstance().getAllLoads(structuralElement);
    for(int i=0; i< loads.size(); i++){
      Load load = (Load) loads.elementAt(i);
      myLoadsPane.addLoad(load);
    }

    //refresh the buttons
    SwingUtilities.invokeLater(refresh);
  }

  /**Add a load to the load pane for display */
  public void addLoad(Load load){
    myLoadsPane.addLoad(load);
  }

  /** Inner class for the loads pane. This pane displays the loads on the
   *  structural element.
   */
  private class LoadsPane extends JPanel{
    /**The list to display the elements */
    JList myJlist = null;
    /**The list model for loads */
    DefaultListModel myLoads = null;

    /**Constructor */
    public LoadsPane(){
      myLoads = new DefaultListModel();

      myJlist = new JList(myLoads) {
        public String getToolTipText(MouseEvent e) {
          int index = locationToIndex(e.getPoint());
          if (-1 < index) {
            Load load = (Load) getModel().getElementAt(index);
            return load.getToolTipText();
          } else {
            return null;
          }
        }
      };
      myJlist.setToolTipText("");
```

```
  myJlist.addListSelectionListener(new MyListSelectionListener());
  myJlist.getModel().addListDataListener(new MyListDataListener());
  myJlist.addKeyListener(new MyKeyListener());
  add(new JScrollPane(myJlist));
}

/** Add a load to the list */
public void addLoad(Load load){
  myLoads.addElement(load);
}

/** Remove the selected load*/
public void removeSelectedLoad(){
  int index = getSelectedIndex();
  if (index != -1)
  myLoads.removeElementAt(index);
}

/** Get the index of the selected element in the list (return -1
    if no element is selected */
public int getSelectedIndex(){
  return myJlist.getSelectedIndex();
}

/** Get the load that is selected. Return null if no load is
  selected.*/
public Load getSelection(){
  int index = getSelectedIndex();
  if (index == -1)
  return null;
  else{
  Object load = myLoads.getElementAt(index);
  Assert.assert(load instanceof Load);
  return (Load) load;
  }
}

/**
 * Inner class --- list selection listener. Changes the button
 * states according to the selection of the element in the function
 * list.
 */
private class MyListSelectionListener implements ListSelectionListener{
  public void valueChanged (ListSelectionEvent evt) {
    //This method is called twice --- due to a mouse click *and*
    //mouse release.  Hence we only refresh for the mouse release
    if(!evt.getValueIsAdjusting()){
      SwingUtilities.invokeLater(refresh);
    }
  }
}

/**
 * Inner class --- list data listener. Refreshes the button states.
 */
class MyListDataListener implements ListDataListener {
  public void contentsChanged(ListDataEvent e) {
```

```
        SwingUtilities.invokeLater(refresh);
      }
      public void intervalAdded(ListDataEvent e) {
        SwingUtilities.invokeLater(refresh);
      }
      public void intervalRemoved(ListDataEvent e) {
        SwingUtilities.invokeLater(refresh);
      }
    }

    /**
     * Inner class --- key event listener. Handles key event typed
     * into the list pane.
     */
    class MyKeyListener extends KeyAdapter {
      /** Handle the key typed event from the function pane. */
      public void keyPressed(KeyEvent e) {
        //if the delete key is pressed then delete the element
        if((e.getKeyCode() == KeyEvent.VK_DELETE) &&
           myButtonsPane.getDeleteButton().isEnabled())
          myDeleteAction.actionPerformed(null);
      }
    }
}

/** Inner class for the buttons pane. This pane displays the Add,
 *  Delete, and Done buttons.
 */
private class ButtonsPane extends JPanel{
  /** New load button */
  JButton myNewLoadBtn = null;
  /** Delete Button */
  JButton myDeleteBtn = null;
  /** Delete Button */
  JButton myDoneBtn = null;

  /**Constructor */
  public ButtonsPane(){
    // create the buttons and register the actions
    myNewLoadBtn = new JButton("New Load");
    myNewLoadBtn.addActionListener(myNewLoadAction);
    myNewLoadBtn.setToolTipText("Click to add a new load.");

    myDeleteBtn = new JButton("Delete Load");
    myDeleteBtn.addActionListener(myDeleteAction);
    myDeleteBtn.setToolTipText("Click to delete the load.");
    myDeleteBtn.setEnabled(false);

    myDoneBtn = new JButton("Done");
    myDoneBtn.addActionListener(myDoneAction);
    myDoneBtn.setToolTipText("Click when finished.");

    // add the buttons
    setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
    setBorder(myPaneEdgeBdr);
    add(myNewLoadBtn);
    add(Box.createRigidArea(new Dimension(5, 0)));
```

```
    add(myDeleteBtn);
    add(Box.createRigidArea(new Dimension(5, 0)));
    add(myDoneBtn);
  }

  /**Get the delete button
   */
  public JButton getDeleteButton(){
    return myDeleteBtn;
  }

  /**
   * Refresh the presentation state of the buttons.
   */
  public void refresh() {
    if (myLoadsPane.getSelectedIndex() == -1){
    myDeleteBtn.setEnabled(false);
    return;
    }
    myDeleteBtn.setEnabled(true);
  }
}

/**Inner class for the new load dialog */
class NewLoadDialog extends JDialog{
  /** The day the load is to be added to the element */
  int myDay  = -1;
  /** The weight of the load */
  float myWeight  = -1;
  /**The parent dialog */
  LoadsDialog myParent = null;
  /** The pane that displays the cancel and finish buttons*/
  JPanel myCancelOkPane = null;
  /**The label for the day the load is to be added*/
  JLabel dayLabel = null;
  /**The field for day the load is to be added*/
  JTextField dayField = null;
 /**The label for weight of the load*/
  JLabel weightLabel = null;
 /**The field for weight of the load*/
  JTextField weightField = null;
  /** The content pane for this dialog */
  Container myContentPane = null;

  /**Constructor */
  public NewLoadDialog(LoadsDialog parent){
    super(parent,true);
    myParent = parent;
    myContentPane = getContentPane();

    JPanel dayPanel = new JPanel();
    dayLabel = new JLabel("Day Added");
    dayField = new JTextField(4);
    dayPanel.add(dayLabel);
    dayPanel.add(dayField);

    JPanel weightPanel = new JPanel();
```

```
  weightLabel = new JLabel("Weight");
  weightField = new JTextField(4);
  weightPanel.add(weightLabel);
  weightPanel.add(weightField);

  myContentPane.setLayout(new BoxLayout(myContentPane,BoxLayout.Y_AXIS));
  myContentPane.add(dayPanel);
  myContentPane.add(Box.createRigidArea(new Dimension(0, 10)));
  myContentPane.add(weightPanel);
  myContentPane.add(Box.createRigidArea(new Dimension(0, 10)));
  myCancelOkPane = new CancelOkPane();
  myContentPane.add(myCancelOkPane);
}

/**Parse the input typed by the user. Displays an error message if
 *   the input is not valid
 */
public void parseInput(){
  try{
  myDay = Integer.parseInt(dayField.getText());
  if (myDay < myStructuralElement.getDayConstructed())
    throw new NumberFormatException();
  }
  catch (NumberFormatException e){
  JOptionPane.showMessageDialog(this,
                                "The day is not valid",
                          "Error",
                                JOptionPane.ERROR_MESSAGE);
  return;
  }

  try{
  myWeight = Float.parseFloat(weightField.getText());
  if (myWeight <= 0)
    throw new NumberFormatException();
  }
  catch (NumberFormatException e){
  JOptionPane.showMessageDialog(this,
                                "The weight is not valid",
                          "Error",
                                JOptionPane.ERROR_MESSAGE);
  return;
  }
}

/**
 * Inner class that defines the cancel action
 */
private class CancelAction extends AbstractAction {
  public void actionPerformed(ActionEvent evt) {
  setVisible(false);
  }
}

/**
 * Inner class that defines the ok action
 */
```

```
  private class OkAction extends AbstractAction {
    public void actionPerformed(ActionEvent evt) {
    try{
      NewLoadDialog.this.parseInput();
      if (myDay > -1 && myWeight > -1){
        Load load = new Load(myStructuralElement, myDay, myWeight,true);
        Structure.getInstance().addLoad(load,myDay);
        myParent.addLoad(load);
        setVisible(false);
      }
    }
    catch(InvalidLoadException e){
      JOptionPane.showMessageDialog(LoadsDialog.this,
                              "Cannot add load",
                              "Error",
                              JOptionPane.ERROR_MESSAGE);
    }
    catch(InvalidParameterException e){
      System.err.println(e.getMessage());
      System.exit(-1);
    }
    }
  }

  /**
   * The cancel/ok pane. Displays the cancel and ok
   * buttons.
   */
  private class CancelOkPane extends JPanel{
    /** Ok Button */
    JButton myOkBtn = null;
    /** Cancel Button */
    JButton myCancelBtn = null;

    /**Constructor*/
    public CancelOkPane(){
    // create the buttons and register the actions
    myCancelBtn = new JButton("Cancel");
    myCancelBtn.addActionListener(new CancelAction());
    myCancelBtn.setToolTipText("Click to cancel.");
    myOkBtn = new JButton("Ok");
    myOkBtn.addActionListener(new OkAction());
    myOkBtn.setToolTipText("Click to finish.");

    //add the buttons to the pane
    BoxLayout layout = new BoxLayout(this, BoxLayout.X_AXIS);
    setLayout(layout);
    setBorder(myPaneEdgeBdr);
    add(Box.createHorizontalGlue());
    add(myCancelBtn);
    add(Box.createRigidArea(new Dimension(10, 0)));
    add(myOkBtn);
    }
  }
}

/**
```

```
   * Inner class that defines the action to add a new load to the list of
loads
   */
  private class NewLoadAction extends AbstractAction {
    public void actionPerformed(ActionEvent evt) {
      NewLoadDialog dialog = new NewLoadDialog(LoadsDialog.this);
      dialog.pack();
      dialog.setVisible(true);
    }
  }


  /**
   * Inner class that defines the actions performed to delete a load
   * from the list of loads
   */
  private class DeleteAction extends AbstractAction {
    public void actionPerformed(ActionEvent evt) {
      Structure.getInstance().removeLoad(myLoadsPane.getSelection());
      myLoadsPane.removeSelectedLoad();
    }
  }

  /**
   * Inner class that defines the done action
   */
  private class DoneAction extends AbstractAction {
    public void actionPerformed(ActionEvent evt) {
      setVisible(false);
    }
  }
}
```

```
/*
 * @(#)Material.java
 *
 * History:  1999-2000: Written by M Gardner and C. Dalton
 *           Oct 2001:  G.Lewis added comments, deleted obsolete code,
 *                and changed variable names.
 *
 * Copyright (C) 2001 Mark Gardner
 */

package structuralAnalyser.material;

import      java.io.*;
import structuralAnalyser.*;

public class Material implements Serializable
{
  /** The kind of material e.g. B3Concrete */
  protected String myKind;
  /** A unique id for the material */
  protected String myId;

  /** Construct the material with a given id
   * @param id the identifier for the material
   */
  public Material(String id, String kind)
  {
    myId = id;
    myKind = kind;
  }
  /** Get the id of the material */
  public String getId()
  {
    return myId;
  }
  /** Get the kind of the material */
  public String getKind()
  {
    return myKind;
  }
  /** Get a string representation of this material */
  public String toString()
  {
    return myId + " (" + myKind + ")";
  }
}
```

```
/*
 * @(#)Metal.java
 *
 * History:  1999-2000: Written by M Gardner and C. Dalton
 *           Oct 2001:  G.Lewis added comments, deleted obsolete code,
 *                and changed variable names.
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser.material;

import java.io.*;
import structuralAnalyser.*;

public class Metal extends Material implements Serializable
{
  protected float         myYield;
  protected float         myDensity;
  protected float         myElasticMod;
  protected float         myThermalExp;
  protected float         myPoisson;

  public Metal(String id, float yield, float density, float elasticMod,
               float thermalExp, float poisson) throws InvalidParameterException
  {
    super(id, "Metal");
    setYield(yield);
    setDensity(density);
    setElasticMod(elasticMod);
    setThermalExp(thermalExp);
    setPoisson(poisson);
  }

  // Accessor functions

  public float getYield()
  {
    return myYield;
  }

  public void setYield(float yield) throws InvalidParameterException
  {
    if (yield <= 0)
      throw new InvalidParameterException("The yield must be greater than 0");
    myYield = yield;
  }

  public float getDensity()
  {
    return myDensity;
  }


  public void setDensity(float density) throws InvalidParameterException
  {
```

```
    if (density <= 1000)
      throw new InvalidParameterException("The density must be greater than
1000.");
    myDensity = density;
  }

  public float getElasticMod()
  {
    return myElasticMod;
  }

  public void setElasticMod(float elasticMod) throws InvalidParameterException
  {
    if (elasticMod <= 0)
      throw new InvalidParameterException("The elastic mod  must be greater
than 0.");
    myElasticMod = elasticMod;
  }

  public float getThermalExp()
  {
    return myThermalExp;
  }

  public void setThermalExp(float thermalExp) throws InvalidParameterException
  {
    if (thermalExp <= 0)
      throw new InvalidParameterException("The thermal exp must be greater
than 0.");
    myThermalExp = thermalExp;
  }


  public float getPoisson()
  {
    return myPoisson;
  }

  public void setPoisson(float poisson) throws InvalidParameterException
  {
    if (poisson <= 0)
      throw new InvalidParameterException("The poisson must be greater than
0.");
    myPoisson = poisson;
  }

}
```

```
/*
 * @(#) NewElementDialog.java
 *
 * History:  Oct 2001: Written by M. Gardner
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */
package structuralAnalyser;

import javax.swing.*;
import javax.swing.border.*;
import javax.accessibility.*;

import java.awt.*;
import java.awt.event.*;

import structuralAnalyser.structuralElement.*;
import structuralAnalyser.material.*;

public class NewElementDialog extends JDialog {
  /** The element defined by the user */
  StructuralElement structuralElement = null;
  /**The day the element was constructed*/
  int dayConstructed = -1;
  /**The starting coorindate of the element */
  Coordinate startCoordinate = null;

  /**The parent frame */
  AnalyserFrame myParent = null;

  /** The different elements that can be constructed */
  private String[] elementStrings = { " Column ", " Beam ",
                                      " Floor ", " Wall "};

  /** Panel for entering column details */
  ColumnPanel myColumnPanel = null;
  /** Panel for entering beam details */
  BeamPanel myBeamPanel = null;
  /** Panel for entering floor details */
  FloorPanel myFloorPanel = null;
  /** Panel for entering wall details */
  WallPanel myWallPanel = null;
  /** The content pane for this dialog */
  Container contentPane = null;
  /** The panel that is currently being displayed */
  StructuralElementPanel currentPanel = null;
  /** The panel that holds the element panel*/
  JPanel myElementContainerPanel = null;
  /** The pane that displays the cancel and finish buttons*/
  JPanel myCancelFinishPane = null;

  /** Finish Action */
  Action myFinishAction = new FinishAction();
  /** Cancel Action */
  Action myCancelAction = new CancelAction();
```

```
  /**A border that puts 10 extra pixels around a pane.*/
  Border myPaneEdgeBdr = BorderFactory.createEmptyBorder(10,10,10,10);

  /** Constructor
   * @param parent the parent frame */
  public NewElementDialog(AnalyserFrame parent)    {
    super(parent,"Structural Element Editor",true);
    contentPane = getContentPane();
    myParent = parent;

    //Set up the column panel
    myColumnPanel  = new ColumnPanel();
    myColumnPanel.setPreferredSize(new Dimension(300, 310));
    myColumnPanel.setOpaque(true);
    myColumnPanel.setForeground(Color.black);

    //Set up the beam panel
    myBeamPanel  = new BeamPanel();
    myBeamPanel.setPreferredSize(new Dimension(300, 310));
    myBeamPanel.setOpaque(true);
    myBeamPanel.setForeground(Color.black);

    //Set up the floor panel
    myFloorPanel  = new FloorPanel();
    myFloorPanel.setPreferredSize(new Dimension(300, 310));
    myFloorPanel.setOpaque(true);
    myFloorPanel.setForeground(Color.black);

    //Set up the wall panel
    myWallPanel  = new WallPanel();
    myWallPanel.setPreferredSize(new Dimension(300, 310));
    myWallPanel.setOpaque(true);
    myWallPanel.setForeground(Color.black);

    //Add control pane and panel to frame.
    contentPane.setLayout(new BoxLayout(contentPane,
                                BoxLayout.Y_AXIS));
    contentPane.add(Box.createRigidArea(new Dimension(0, 10)));
    contentPane.add(createControlPanel());
    contentPane.add(Box.createRigidArea(new Dimension(0, 10)));
    myElementContainerPanel = new JPanel();

myElementContainerPanel.setBorder(BorderFactory.createLineBorder(Color.black))
;
    myElementContainerPanel.add(myColumnPanel);
    contentPane.add(myElementContainerPanel);
    currentPanel = myColumnPanel;
    //add the cancel and finish buttons
    myCancelFinishPane = new CancelFinishPane();
    contentPane.add(Box.createRigidArea(new Dimension(0, 10)));
    contentPane.add(myCancelFinishPane);
  }

  /**Abstract panel for entering structural elements details */
  abstract class StructuralElementPanel extends JPanel{
    /**The day the element was constructed */
    private JLabel dayLabel = null;
```

```
private JTextField dayField = null;
/**The material of the element */
protected Material material = null;
private JLabel materialLabel = null;
private JComboBox materialBox = null;
/**The starting position of the element */
private JLabel startCoordLabel = null;
private JTextField startCoordField = null;

/**Constructor */
public StructuralElementPanel(){
  setLayout(new GridLayout(0,1));
  JPanel dayPanel = new JPanel();
  dayLabel = new JLabel("Day");
  dayField = new JTextField(4);
  dayPanel.add(dayLabel);
  dayPanel.add(dayField);

  JPanel startCoordPanel = new JPanel();
  startCoordLabel = new JLabel("Starting Coordinate (x,y,z)");
  startCoordField = new JTextField(10);
  startCoordPanel.add(startCoordLabel);
  startCoordPanel.add(startCoordField);

  JPanel materialPanel = new JPanel();
  materialLabel = new JLabel("Material");
  materialBox = new JComboBox(Structure.getInstance().getMaterials());
  materialPanel.add(materialLabel);
  materialPanel.add(materialBox);

  add(dayPanel);
  add(materialPanel);
  add(startCoordPanel);
}

public boolean parseInput(){
  try{
  dayConstructed = Integer.parseInt(dayField.getText());
  if (dayConstructed < 0)
    throw new NumberFormatException();
  }
  catch (NumberFormatException e){
  JOptionPane.showMessageDialog(this,
                                "The day constructed is not valid",
                          "Error",
                                JOptionPane.ERROR_MESSAGE);
  return false;
  }

  material = (Material) materialBox.getSelectedItem();
  if (material == null){
  JOptionPane.showMessageDialog(this,
                          "You must specify a material.",
                          "Error",
                          JOptionPane.ERROR_MESSAGE);
  return false;
  }
```

```
      try{
      startCoordinate =
Coordinate.parseCoordinate(startCoordField.getText());
      }
      catch(InvalidParameterException e){
      JOptionPane.showMessageDialog(this,
                                "Start Coordinate error: " +
                                e.getMessage(),
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
      return false;
      }
      return true;
    }
  }


  /**Panel for entering in a columns details */
  class ColumnPanel extends StructuralElementPanel{
    /**The length of the column */
    private JLabel lengthLabel = null;
    private JTextField lengthField = null;
    /**The section of the column */
    private Section section = null;
    private JLabel sectionLabel = null;
    private JComboBox sectionBox = null;

    /**Constructor */
    public ColumnPanel()
    {
      JPanel lengthPanel = new JPanel();
      lengthLabel = new JLabel("Length");
      lengthField = new JTextField(4);
      lengthPanel.add(lengthLabel);
      lengthPanel.add(lengthField);

      JPanel sectionPanel = new JPanel();
      sectionLabel = new JLabel("Section");
      sectionBox = new JComboBox(Structure.getInstance().getSections());
      sectionPanel.add(sectionLabel);
      sectionPanel.add(sectionBox);

      add(lengthPanel);
      add(sectionPanel);
    }

    /**Parse the input typed by the user. Returns true if the input
     *   was successfully parsed.
     */
    public boolean parseInput(){
      if (!super.parseInput())
      return false;

      float length;
      try{
      length = Float.parseFloat(lengthField.getText());
```

```
      if (length <= 0)
        throw new NumberFormatException();
      }
      catch (NumberFormatException e){
      JOptionPane.showMessageDialog(this,
                                "The length of the column is not valid",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
      return false;
      }

      section = (Section) sectionBox.getSelectedItem();
      if (section == null){
      JOptionPane.showMessageDialog(this,
                                "You must specify a section for the column",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
      return false;
      }


      try{
      structuralElement = new
Column(Structure.getInstance().getNextUniqueId(), material,
                                 length,section);
      }
      catch (InvalidParameterException e){
      System.err.println(e);
      }
      return true;
    }
  }

  /**Panel for entering in beam details */
  class BeamPanel extends StructuralElementPanel{
    /**The section of the beam */
    private Section section = null;
    private JLabel sectionLabel = null;
    private JComboBox sectionBox = null;

    /**The ending position of the beam */
    private Coordinate endCoordinate  = null;
    private JLabel endCoordLabel = null;
    private JTextField endCoordField = null;

    /**The fixed position of the beam */
    private Coordinate fixedCoordinate  = null;
    private JLabel fixedCoordLabel = null;
    private JTextField fixedCoordField = null;

    public BeamPanel()
    {
      JPanel sectionPanel = new JPanel();
      sectionLabel = new JLabel("Section");
      sectionBox = new JComboBox(Structure.getInstance().getSections());
      sectionPanel.add(sectionLabel);
      sectionPanel.add(sectionBox);
```

```
        JPanel endCoordPanel = new JPanel();
        endCoordLabel = new JLabel("Ending Coordinate (x,y,z)");
        endCoordField = new JTextField(10);
        endCoordPanel.add(endCoordLabel);
        endCoordPanel.add(endCoordField);

        JPanel fixedCoordPanel = new JPanel();
        fixedCoordLabel = new JLabel("Fixed Coordinate (x,y,z)");
        fixedCoordField = new JTextField(10);
        fixedCoordPanel.add(fixedCoordLabel);
        fixedCoordPanel.add(fixedCoordField);

        add(sectionPanel);
        add(endCoordPanel);
        add(fixedCoordPanel);
      }

      /**Parse the input typed by the user. Returns true if the input
       *   was successfully parsed.
       */
      public boolean parseInput(){
        if (!super.parseInput())
        return false;

        section = (Section) sectionBox.getSelectedItem();
        if (section == null){
        JOptionPane.showMessageDialog(this,
                                  "You must specify a section for the beam",
                                  "Error",
                                  JOptionPane.ERROR_MESSAGE);
        return false;
        }

        try{
        endCoordinate =  Coordinate.parseCoordinate(endCoordField.getText());
        }
        catch(InvalidParameterException e){
        JOptionPane.showMessageDialog(this,
                                  "End Coordinate error: " +
                                  e.getMessage(),
                                  "Error",
                                  JOptionPane.ERROR_MESSAGE);
        return false;
        }

        try{
        fixedCoordinate =
Coordinate.parseCoordinate(fixedCoordField.getText());
        }
        catch(InvalidParameterException e){
        JOptionPane.showMessageDialog(this,
                                  "Fixed Coordinate error: " +
                                  e.getMessage(),
                                  "Error",
                                  JOptionPane.ERROR_MESSAGE);
        return false;
```

```
      }

      try{
      structuralElement = new Beam(Structure.getInstance().getNextUniqueId(),
material,
                              section,endCoordinate,fixedCoordinate);
      }
      catch (InvalidParameterException e){
      System.err.println(e);
      }
      return true;
    }
  }

  /**Panel for entering in floor details */
  class FloorPanel extends StructuralElementPanel{
    /**The thickness of the floor*/
    private JLabel thicknessLabel = null;
    private JTextField thicknessField = null;

    /**The ending position of the floor */
    private Coordinate endCoordinate  = null;
    private JLabel endCoordLabel = null;
    private JTextField endCoordField = null;

    public FloorPanel()
    {
      JPanel thicknessPanel = new JPanel();
      thicknessLabel = new JLabel("Thickness");
      thicknessField = new JTextField(4);
      thicknessPanel.add(thicknessLabel);
      thicknessPanel.add(thicknessField);

      JPanel endCoordPanel = new JPanel();
      endCoordLabel = new JLabel("Ending Coordinate (x,y,z)");
      endCoordField = new JTextField(10);
      endCoordPanel.add(endCoordLabel);
      endCoordPanel.add(endCoordField);

      add(thicknessPanel);
      add(endCoordPanel);
    }

    /**Parse the input typed by the user. Returns true if the input
     * was successfully parsed.
     */
    public boolean parseInput(){
      if (!super.parseInput())
      return false;

      float thickness;
      try{
      thickness = Float.parseFloat(thicknessField.getText());
      if (thickness <= 0)
        throw new NumberFormatException();
      }
      catch (NumberFormatException e){
```

```
    JOptionPane.showMessageDialog(this,
                                  "The thickness of the floor is not valid",
                                  "Error",
                                  JOptionPane.ERROR_MESSAGE);
    return false;
    }
    try{
    endCoordinate =  Coordinate.parseCoordinate(endCoordField.getText());
    }
    catch(InvalidParameterException e){
    JOptionPane.showMessageDialog(this,
                                  "End Coordinate error: " +
                                  e.getMessage(),
                                  "Error",
                                  JOptionPane.ERROR_MESSAGE);
    return false;
    }

    try{
    structuralElement = new Floor(Structure.getInstance().getNextUniqueId(),
                                  material,thickness,endCoordinate);
    }
    catch (InvalidParameterException e){
    System.err.println(e);
    }
    return true;
  }
}

/**Panel for entering in wall details */
class WallPanel extends StructuralElementPanel{
  public WallPanel()
  {
    JLabel lbl = new JLabel("Wall specific details to go here.");
    add(lbl);
  }
}

//Create the control pane for the top of the frame.
private JPanel createControlPanel() {
  final JComboBox layerList = new JComboBox(elementStrings);
  layerList.setSelectedIndex(0);
  layerList.addActionListener(new ActionListener () {
    public void actionPerformed(ActionEvent e) {
      int index = layerList.getSelectedIndex();
      JPanel oldPanel = currentPanel;
      if (index == 0)
        currentPanel = myColumnPanel;
      else if (index == 1)
        currentPanel = myBeamPanel;
      else if (index == 2)
        currentPanel = myFloorPanel;
      else if (index == 3)
        currentPanel = myWallPanel;
      if (oldPanel != currentPanel){
        oldPanel.setVisible(false);
        myElementContainerPanel.add(currentPanel);
```

```
            currentPanel.setVisible(true);
          }
        }
        });

    JPanel controls = new JPanel();
    controls.add(layerList);
    controls.setBorder(BorderFactory.createTitledBorder("Choose the structural
element."));
    return controls;
  }


  /**
   * The cancel/finish pane. Displays the cancel and finish
   * buttons.
   */
  private class CancelFinishPane extends JPanel{
    /** Finish Button */
    JButton myFinishBtn = null;
    /** Cancel Button */
    JButton myCancelBtn = null;

    /**Constructor*/
    public CancelFinishPane(){
      // create the buttons and register the actions
      myCancelBtn = new JButton("Cancel");
      myCancelBtn.addActionListener(myCancelAction);
      myCancelBtn.setToolTipText("Click to cancel.");
      myFinishBtn = new JButton("Finish");
      myFinishBtn.addActionListener(myFinishAction);
      myFinishBtn.setToolTipText("Click to finish.");

      //add the buttons to the pane
      BoxLayout layout = new BoxLayout(this, BoxLayout.X_AXIS);
      setLayout(layout);
      setBorder(myPaneEdgeBdr);
      add(Box.createHorizontalGlue());
      add(myCancelBtn);
      add(Box.createRigidArea(new Dimension(10, 0)));
      add(myFinishBtn);
    }
  }

  /**
   * Inner class that defines the cancel action
   */
  private class CancelAction extends AbstractAction {
    public void actionPerformed(ActionEvent evt) {
      setVisible(false);
    }
  }


  /**
   * Inner class that defines the finish action
   */
```

```
  private class FinishAction extends AbstractAction {
    public void actionPerformed(ActionEvent evt) {
      if (currentPanel.parseInput()){
      Assert.assert(structuralElement!= null &&
                  dayConstructed >=0 &&
                  startCoordinate != null);
      try{

Structure.getInstance().addElement(structuralElement,dayConstructed,startCoord
inate);
        myParent.addElement(structuralElement);
        setVisible(false);
      }
      catch(InvalidPositionException e){
        JOptionPane.showMessageDialog(NewElementDialog.this,
                                "The element cannot be added at that position.",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
      }
      catch(InvalidParameterException e){
        JOptionPane.showMessageDialog(NewElementDialog.this,
                                "The element cannot be added to the structure",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
      }
      }
    }
  }
}
```

```
/*
 * @(#) NewMaterialDialog.java
 *
 * History:   Mar 2002: Written by M Gardner.
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

import javax.swing.*;
import javax.swing.border.*;
import javax.accessibility.*;

import java.util.*;
import java.awt.*;
import java.awt.event.*;

import structuralAnalyser.structuralElement.*;
import structuralAnalyser.material.*;

public class NewMaterialDialog extends JDialog {
  /**The instance of the material created by the user */
  Material myMaterial = null;

  /**The parent frame */
  AnalyserFrame myParent = null;

  /** The different elements that can be constructed */
  private Vector materialStrings = null;

  /** Panel for entering column details */
  ACIConcretePanel myACIConcretePanel = null;
  /** Panel for entering B3Concrete details */
  B3ConcretePanel myB3ConcretePanel = null;
  /** Panel for entering metal details */
  MetalPanel myMetalPanel = null;
  /** Panel for entering timber details */
  TimberPanel myTimberPanel = null;
  /** The content pane for this dialog */
  Container contentPane = null;
  /** The panel that is currently being displayed */
  MaterialPanel currentPanel = null;
  /** The panel that holds the element panel*/
  JPanel myElementContainerPanel = null;
  /** The pane that displays the cancel and finish buttons*/
  JPanel myCancelFinishPane = null;

  /** Finish Action */
  Action myFinishAction = new FinishAction();
  /** Cancel Action */
  Action myCancelAction = new CancelAction();

  /**A border that puts 10 extra pixels around a pane.*/
  Border myPaneEdgeBdr = BorderFactory.createEmptyBorder(10,10,10,10);
```

```
  /** Constructor
   * @param parent the parent frame */
  public NewMaterialDialog(AnalyserFrame parent)    {
    super(parent,"New Material Editor",true);
    contentPane = getContentPane();
    myParent = parent;

    //Set up the column panel
    myACIConcretePanel  = new ACIConcretePanel();
    myACIConcretePanel.setOpaque(true);
    myACIConcretePanel.setForeground(Color.black);

    //Set up the B3Concrete panel
    myB3ConcretePanel  = new B3ConcretePanel();
    myB3ConcretePanel.setOpaque(true);
    myB3ConcretePanel.setForeground(Color.black);

    //Set up the metal panel
    myMetalPanel  = new MetalPanel();
    myMetalPanel.setOpaque(true);
    myMetalPanel.setForeground(Color.black);

    //Set up the timber panel
    myTimberPanel  = new TimberPanel();
    myTimberPanel.setOpaque(true);
    myTimberPanel.setForeground(Color.black);

    //Add control pane and panel to frame.
    contentPane.setLayout(new BoxLayout(contentPane,
                                BoxLayout.Y_AXIS));
    contentPane.add(Box.createRigidArea(new Dimension(0, 10)));
    contentPane.add(createControlPanel());
    contentPane.add(Box.createRigidArea(new Dimension(0, 10)));
    myElementContainerPanel = new JPanel();

myElementContainerPanel.setBorder(BorderFactory.createLineBorder(Color.black))
;

    myElementContainerPanel.add(myMetalPanel);
    contentPane.add(myElementContainerPanel);
    currentPanel = myMetalPanel;
    //add the cancel and finish buttons
    myCancelFinishPane = new CancelFinishPane();
    contentPane.add(Box.createRigidArea(new Dimension(0, 10)));
    contentPane.add(myCancelFinishPane);
  }

  public abstract class MaterialPanel extends JPanel{
    /**The id for the material */
    protected String id = null;
    private JLabel idLabel = null;
    private JTextField idField = null;

    /**Constructor */
    public MaterialPanel(){
      setLayout(new GridLayout(0,2));
      JPanel idPanel = new JPanel();
```

```
        idLabel = new JLabel("Name");
        idField = new JTextField(10);
        idPanel.add(idLabel);
        idPanel.add(idField);

        add(idPanel);
      }
      /**Parse the input typed by the user. Return true if the input was
       *   successfully parsed.
       */
      public boolean parseInput(){
        id = idField.getText().trim();
        if (id.equals("") || id == null){
        JOptionPane.showMessageDialog(NewMaterialDialog.this,
                                      "You must enter a name for the material",
                    "Error",
                                      JOptionPane.ERROR_MESSAGE);
        id = null;
        return false;
        }
        //check that the id is unqiue
        Vector materials = Structure.getInstance().getMaterials();
        for (int i = 0; i < materials.size(); i++){
        Material material = (Material) materials.elementAt(i);
        if (material.getId().equals(id)){
          JOptionPane.showMessageDialog(NewMaterialDialog.this,
                                      "This name has already been used for a material
in this structure.",
                                      "Error",
                                      JOptionPane.ERROR_MESSAGE);
          id = null;
          return false;
        }
        }
        return true;
      }
    }



    /**Panel for entering in concrete details */
    abstract class ConcretePanel extends MaterialPanel{
      /** Conc28 day for the concrete */
      protected float conc28day;
      private JLabel conc28dayLabel = null;
      private JTextField conc28dayField = null;
      /** Density for the concrete */
      protected float density;
      private JLabel densityLabel = null;
      private JTextField densityField = null;
      /** elasticMod for the concrete */
      protected float elasticMod;
      private JLabel elasticModLabel = null;
      private JTextField elasticModField = null;
      /** thermalExp for the concrete */
      protected float thermalExp;
      private JLabel thermalExpLabel = null;
```

```
private JTextField thermalExpField = null;
/** humidity for the concrete */
protected float humidity;
private JLabel humidityLabel = null;
private JTextField humidityField = null;
/** metalReinforcingArea for the concrete */
protected float metalReinforcingArea;
private JLabel metalReinforcingAreaLabel = null;
private JTextField metalReinforcingAreaField = null;
/** cementType for the concrete */
protected String cementType;
private JLabel cementTypeLabel = null;
private JTextField cementTypeField = null;
/** cementContent for the concrete */
protected float cementContent;
private JLabel cementContentLabel = null;
private JTextField cementContentField = null;
/** waterCementRatio for the concrete */
protected float waterCementRatio;
private JLabel waterCementRatioLabel = null;
private JTextField waterCementRatioField = null;
/** addedMetal for the concrete */
protected Metal addedMetal;
private JLabel addedMetalLabel = null;
private JComboBox addedMetalBox = null;

/**Constructor */
public ConcretePanel()
{
  JPanel conc28dayPanel = new JPanel();
  conc28dayLabel = new JLabel("28 Day");
  conc28dayField = new JTextField(4);
  conc28dayPanel.add(conc28dayLabel);
  conc28dayPanel.add(conc28dayField);

  JPanel densityPanel = new JPanel();
  densityLabel = new JLabel("Density");
  densityField = new JTextField(4);
  densityPanel.add(densityLabel);
  densityPanel.add(densityField);

  JPanel elasticModPanel = new JPanel();
  elasticModLabel = new JLabel("Elasitic Mod");
  elasticModField = new JTextField(4);
  elasticModPanel.add(elasticModLabel);
  elasticModPanel.add(elasticModField);

  JPanel thermalExpPanel = new JPanel();
  thermalExpLabel = new JLabel("Basic Shrinkage");
  thermalExpField = new JTextField(4);
  thermalExpPanel.add(thermalExpLabel);
  thermalExpPanel.add(thermalExpField);

  JPanel humidityPanel = new JPanel();
  humidityLabel = new JLabel("Humidity");
  humidityField = new JTextField(4);
  humidityPanel.add(humidityLabel);
```

```
    humidityPanel.add(humidityField);

    JPanel metalReinforcingAreaPanel = new JPanel();
    metalReinforcingAreaLabel = new JLabel("Metal Reinforcing Area");
    metalReinforcingAreaField = new JTextField(4);
    metalReinforcingAreaPanel.add(metalReinforcingAreaLabel);
    metalReinforcingAreaPanel.add(metalReinforcingAreaField);

    JPanel cementTypePanel = new JPanel();
    cementTypeLabel = new JLabel("Cement Type");
    cementTypeField = new JTextField(4);
    cementTypePanel.add(cementTypeLabel);
    cementTypePanel.add(cementTypeField);

    JPanel cementContentPanel = new JPanel();
    cementContentLabel = new JLabel("Cement Content");
    cementContentField = new JTextField(4);
    cementContentPanel.add(cementContentLabel);
    cementContentPanel.add(cementContentField);

    JPanel waterCementRatioPanel = new JPanel();
    waterCementRatioLabel = new JLabel("Water-Cement Ratio");
    waterCementRatioField = new JTextField(4);
    waterCementRatioPanel.add(waterCementRatioLabel);
    waterCementRatioPanel.add(waterCementRatioField);

    JPanel addedMetalPanel = new JPanel();
    addedMetalLabel = new JLabel("Added Metal");
    Vector availableMetals = new Vector();
    Vector materials = Structure.getInstance().getMaterials();
    for (int i = 0; i < materials.size(); i++){
    if (materials.elementAt(i) instanceof Metal)
      availableMetals.add(materials.elementAt(i));
    }
    addedMetalBox = new JComboBox(availableMetals);
    addedMetalPanel.add(addedMetalLabel);
    addedMetalPanel.add(addedMetalBox);

    add(conc28dayPanel);
    add(densityPanel);
    add(elasticModPanel);
    add(thermalExpPanel);
    add(humidityPanel);
    add(metalReinforcingAreaPanel);
    add(cementTypePanel);
    add(cementContentPanel);
    add(waterCementRatioPanel);
    add(addedMetalPanel);
  }

  /**Parse the input typed by the user. Return true if the input was
   * successfully parsed.
   */
  public boolean parseInput(){
    if (!super.parseInput())
    return false;
    try{
```

```
conc28day = Float.parseFloat(conc28dayField.getText());
if (conc28day <= 10)
  throw new NumberFormatException();
}
catch(NumberFormatException e){
JOptionPane.showMessageDialog(this,
                        "conc 28 day is not valid.",
                        "Error",
                        JOptionPane.ERROR_MESSAGE);
return false;
}

try{
density = Float.parseFloat(densityField.getText());
if (density <= 1000)
  throw new NumberFormatException();
}
catch(NumberFormatException e){
JOptionPane.showMessageDialog(this,
                        "The density is not valid.",
                        "Error",
                        JOptionPane.ERROR_MESSAGE);
return false;
}

try{
elasticMod = Float.parseFloat(elasticModField.getText());
if (elasticMod < 0)
  throw new NumberFormatException();
}
catch(NumberFormatException e){
JOptionPane.showMessageDialog(this,
                        "The elastic mod is not valid.",
                        "Error",
                        JOptionPane.ERROR_MESSAGE);
return false;
}

try{
thermalExp = Float.parseFloat(thermalExpField.getText());
if (thermalExp < 0)
  throw new NumberFormatException();
}
catch(NumberFormatException e){
JOptionPane.showMessageDialog(this,
                        "The thermal exp is not valid.",
                        "Error",
                        JOptionPane.ERROR_MESSAGE);
return false;
}

try{
humidity = Float.parseFloat(humidityField.getText());
if (humidity < 0)
  throw new NumberFormatException();
}
catch(NumberFormatException e){
```

```
        JOptionPane.showMessageDialog(this,
                                "The  humidity is not valid.",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
        return false;
        }

        try{
        metalReinforcingArea =
    Float.parseFloat(metalReinforcingAreaField.getText());
        if (metalReinforcingArea < 0)
          throw new NumberFormatException();
        }
        catch(NumberFormatException e){
        JOptionPane.showMessageDialog(this,
                                "The metal reinforcing area is not valid.",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
        return false;
        }

        try{
        cementType = cementTypeField.getText();
        if (cementType == null) // NEED BETTER TEST FOR CHAR
          throw new NumberFormatException();
        }
        catch(NumberFormatException e){
        JOptionPane.showMessageDialog(this,
                                "The cement type is not valid.",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
        return false;
        }

        try{
        cementContent = Float.parseFloat(cementContentField.getText());
        if (cementContent < 0)//NEED BETTER TEST THAN THIS
          throw new NumberFormatException();
        }
        catch(NumberFormatException e){
        JOptionPane.showMessageDialog(this,
                                "The cement content is not valid.",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
        return false;
        }
        try{
        waterCementRatio = Float.parseFloat(waterCementRatioField.getText());
        if (waterCementRatio < 0)
          throw new NumberFormatException();
        }
        catch(NumberFormatException e){
        JOptionPane.showMessageDialog(this,
                                "The water-cement ratio is not valid.",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
        return false;
```

```
    }

    addedMetal = (Metal) addedMetalBox.getSelectedItem();
    if (addedMetal == null){
    JOptionPane.showMessageDialog(this,
                              "You must select a metal for the concrete.",
                              "Error",
                              JOptionPane.ERROR_MESSAGE);
    return false;
    }

    return true;
  }
}

/**Panel for entering in ACI concrete details */
class ACIConcretePanel extends ConcretePanel{
  /** percentFineAggregate for the concrete */
  protected float percentFineAggregate;
  private JLabel percentFineAggregateLabel = null;
  private JTextField percentFineAggregateField = null;
  /** percentAir for the concrete */
  protected float percentAir;
  private JLabel percentAirLabel = null;
  private JTextField percentAirField = null;
  /** slump for the concrete */
  protected float slump;
  private JLabel slumpLabel = null;
  private JTextField slumpField = null;
  /** curingTime for the concrete */
  protected float curingTime;
  private JLabel curingTimeLabel = null;
  private JTextField curingTimeField = null;

  /**Constructor */
  public ACIConcretePanel()
  {
    JPanel percentFineAggregatePanel = new JPanel();
    percentFineAggregateLabel = new JLabel("Percent Fine Aggregate");
    percentFineAggregateField = new JTextField(4);
    percentFineAggregatePanel.add(percentFineAggregateLabel);
    percentFineAggregatePanel.add(percentFineAggregateField);

    JPanel percentAirPanel = new JPanel();
    percentAirLabel = new JLabel("Percent Air");
    percentAirField = new JTextField(4);
    percentAirPanel.add(percentAirLabel);
    percentAirPanel.add(percentAirField);

    JPanel slumpPanel = new JPanel();
    slumpLabel = new JLabel("Slump");
    slumpField = new JTextField(4);
    slumpPanel.add(slumpLabel);
    slumpPanel.add(slumpField);

    JPanel curingTimePanel = new JPanel();
    curingTimeLabel = new JLabel("Curing Time");
```

```
        curingTimeField = new JTextField(4);
        curingTimePanel.add(curingTimeLabel);
        curingTimePanel.add(curingTimeField);

        add(percentFineAggregatePanel);
        add(percentAirPanel);
        add(slumpPanel);
        add(curingTimePanel);
      }

      /**Parse the input typed by the user. Return true if the input was
       *   successfully parsed. */
      public boolean parseInput(){
        if (!super.parseInput())
        return false;

        try{
        percentFineAggregate =
  Float.parseFloat(percentFineAggregateField.getText());
        if ((percentFineAggregate < 0) || (percentFineAggregate > 100))
          throw new NumberFormatException();
        }
        catch(NumberFormatException e){
        JOptionPane.showMessageDialog(this,
                                "The percent fine aggregate is not valid.",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
        return false;
        }

        try{
        percentAir = Float.parseFloat(percentAirField.getText());
        if ((percentAir < 0) || (percentAir > 100))
          throw new NumberFormatException();
        }
        catch(NumberFormatException e){
        JOptionPane.showMessageDialog(this,
                                "The percent air is not valid.",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
        return false;
        }

        try{
        slump = Float.parseFloat(slumpField.getText());
        }
        catch(NumberFormatException e){
        JOptionPane.showMessageDialog(this,
                                "The slump is not valid.",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
        return false;
        }

        try{
        curingTime = Float.parseFloat(curingTimeField.getText());
        if (curingTime <= 0)
```

```
        throw new NumberFormatException();
      }
      catch(NumberFormatException e){
      JOptionPane.showMessageDialog(this,
                                    "The curing time is not valid.",
                                    "Error",
                                    JOptionPane.ERROR_MESSAGE);
      return false;
      }

      try{
      myMaterial = new ACIConcrete(id, conc28day, density, elasticMod,
                                   thermalExp, humidity,
                                   metalReinforcingArea, cementType,
                                   cementContent, waterCementRatio,
                                   addedMetal, percentFineAggregate,
                                   percentAir, slump, curingTime);
      }
      catch (InvalidParameterException e){
      System.err.println(e.getMessage());
      System.exit(-1);
      }
      return true;
    }
}


/**Panel for entering in B3 concrete details */
class B3ConcretePanel extends ConcretePanel{
  /** water content for the concrete */
  protected float waterContent;
  private JLabel waterContentLabel = null;
  private JTextField waterContentField = null;
  /** aggregate cement ratio for the concrete */
  protected float aggregateCementRatio;
  private JLabel aggregateCementRatioLabel = null;
  private JTextField aggregateCementRatioField = null;
  /** alpha2 for the concrete */
  protected float alpha2;
  private JLabel alpha2Label = null;
  private JTextField alpha2Field = null;
  /** ks for the concrete */
  protected float ks;
  private JLabel ksLabel = null;
  private JTextField ksField = null;

  /**Constructor */
  public B3ConcretePanel()
  {
    JPanel waterContentPanel = new JPanel();
    waterContentLabel = new JLabel("Water Content");
    waterContentField = new JTextField(4);
    waterContentPanel.add(waterContentLabel);
    waterContentPanel.add(waterContentField);

    JPanel aggregateCementRatioPanel = new JPanel();
    aggregateCementRatioLabel = new JLabel("Aggregate Cement Ratio");
```

```
        aggregateCementRatioField = new JTextField(4);
        aggregateCementRatioPanel.add(aggregateCementRatioLabel);
        aggregateCementRatioPanel.add(aggregateCementRatioField);

        JPanel alpha2Panel = new JPanel();
        alpha2Label = new JLabel("Alpha2");
        alpha2Field = new JTextField(4);
        alpha2Panel.add(alpha2Label);
        alpha2Panel.add(alpha2Field);

        JPanel ksPanel = new JPanel();
        ksLabel = new JLabel("KS");
        ksField = new JTextField(4);
        ksPanel.add(ksLabel);
        ksPanel.add(ksField);

        add(waterContentPanel);
        add(aggregateCementRatioPanel);
        add(alpha2Panel);
        add(ksPanel);
    }

    /**Parse the input typed by the user. Return true if the input was
     *   successfully parsed. */
    public boolean parseInput(){
      if (!super.parseInput())
      return false;

      try{
      waterContent = Float.parseFloat(waterContentField.getText());
      if (waterContent < 0)
        throw new NumberFormatException();
      }
      catch(NumberFormatException e){
      JOptionPane.showMessageDialog(this,
                                "The water content is not valid.",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
      return false;
      }

      try{
      aggregateCementRatio =
Float.parseFloat(aggregateCementRatioField.getText());
      if (aggregateCementRatio < 0)
        throw new NumberFormatException();
      }
      catch(NumberFormatException e){
      JOptionPane.showMessageDialog(this,
                                "The percent air is not valid.",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
      return false;
      }

      try{
      alpha2 = Float.parseFloat(alpha2Field.getText());
```

```
        if (alpha2 <= 0)
          throw new NumberFormatException();
        }
        catch(NumberFormatException e){
        JOptionPane.showMessageDialog(this,
                                "Alpha 2 is not valid.",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
        return false;
        }

        try{
        ks = Float.parseFloat(ksField.getText());
        if (ks <= 0)
          throw new NumberFormatException();
        }
        catch(NumberFormatException e){
        JOptionPane.showMessageDialog(this,
                                "ks is not valid.",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
        return false;
        }

        try{
        myMaterial = new B3Concrete(id, conc28day, density, elasticMod,
                                thermalExp, humidity,
                                metalReinforcingArea, cementType,
                                cementContent, waterCementRatio,
                                addedMetal, waterContent,
                                aggregateCementRatio, alpha2, ks);
        }
        catch (InvalidParameterException e){
        System.err.println(e.getMessage());
        System.exit(-1);
        }
        return true;
      }
    }

    /**Panel for entering in metal details */
    class MetalPanel extends MaterialPanel{
      /** Yield  for the metal */
      protected float yield;
      private JLabel yieldLabel = null;
      private JTextField yieldField = null;
      /** Density for the metal */
      protected float density;
      private JLabel densityLabel = null;
      private JTextField densityField = null;
      /** elasticMod for the metal */
      protected float elasticMod;
      private JLabel elasticModLabel = null;
      private JTextField elasticModField = null;
      /** thermalExp for the metal */
      protected float thermalExp;
      private JLabel thermalExpLabel = null;
```

```
private JTextField thermalExpField = null;
/** poisson for the metal */
protected float poisson;
private JLabel poissonLabel = null;
private JTextField poissonField = null;

/**Constructor */
public MetalPanel()
{
  JPanel yieldPanel = new JPanel();
  yieldLabel = new JLabel("Yield");
  yieldField = new JTextField(4);
  yieldPanel.add(yieldLabel);
  yieldPanel.add(yieldField);

  JPanel densityPanel = new JPanel();
  densityLabel = new JLabel("Density");
  densityField = new JTextField(4);
  densityPanel.add(densityLabel);
  densityPanel.add(densityField);

  JPanel elasticModPanel = new JPanel();
  elasticModLabel = new JLabel("Elasitic Mod");
  elasticModField = new JTextField(4);
  elasticModPanel.add(elasticModLabel);
  elasticModPanel.add(elasticModField);

  JPanel thermalExpPanel = new JPanel();
  thermalExpLabel = new JLabel("Thermal Exp");
  thermalExpField = new JTextField(4);
  thermalExpPanel.add(thermalExpLabel);
  thermalExpPanel.add(thermalExpField);

  JPanel poissonPanel = new JPanel();
  poissonLabel = new JLabel("Poisson");
  poissonField = new JTextField(4);
  poissonPanel.add(poissonLabel);
  poissonPanel.add(poissonField);

  add(yieldPanel);
  add(densityPanel);
  add(elasticModPanel);
  add(thermalExpPanel);
  add(poissonPanel);
}

/**Parse the input typed by the user. Return true if the input was
 *   successfully parsed.
 */
public boolean parseInput(){
  if (!super.parseInput())
  return false;
  try{
  yield = Float.parseFloat(yieldField.getText());
  if (yield <= 0)
    throw new NumberFormatException();
  }
```

```
catch(NumberFormatException e){
JOptionPane.showMessageDialog(this,
                        "Yield is not valid.",
                        "Error",
                        JOptionPane.ERROR_MESSAGE);
return false;
}

try{
density = Float.parseFloat(densityField.getText());
if (density <= 1000)
  throw new NumberFormatException();
}
catch(NumberFormatException e){
JOptionPane.showMessageDialog(this,
                        "The density is not valid.",
                        "Error",
                        JOptionPane.ERROR_MESSAGE);
return false;
}

try{
elasticMod = Float.parseFloat(elasticModField.getText());
if (elasticMod < 0)
  throw new NumberFormatException();
}
catch(NumberFormatException e){
JOptionPane.showMessageDialog(this,
                        "The elastic mod is not valid.",
                        "Error",
                        JOptionPane.ERROR_MESSAGE);
return false;
}

try{
thermalExp = Float.parseFloat(thermalExpField.getText());
if (thermalExp < 0)
  throw new NumberFormatException();
}
catch(NumberFormatException e){
JOptionPane.showMessageDialog(this,
                        "The thermal exp is not valid.",
                        "Error",
                        JOptionPane.ERROR_MESSAGE);
return false;
}

try{
poisson = Float.parseFloat(poissonField.getText());
if (poisson < 0)
  throw new NumberFormatException();
}
catch(NumberFormatException e){
JOptionPane.showMessageDialog(this,
                        "The poisson is not valid.",
                        "Error",
                        JOptionPane.ERROR_MESSAGE);
```

```
      return false;
      }

      try{
      myMaterial = new Metal(id,yield,density,elasticMod,thermalExp,poisson);
      }
      catch(InvalidParameterException e){
      System.err.println(e.getMessage());
      System.exit(-1);
      }

      return true;
    }
}

/**Panel for entering in timber details */
class TimberPanel extends MaterialPanel{
  /** Yield  for the timber */
  protected float yield;
  private JLabel yieldLabel = null;
  private JTextField yieldField = null;
  /** Density for the timber */
  protected float density;
  private JLabel densityLabel = null;
  private JTextField densityField = null;
  /** elasticMod for the timber */
  protected float elasticMod;
  private JLabel elasticModLabel = null;
  private JTextField elasticModField = null;

  /**Constructor */
  public TimberPanel()
  {
    JPanel yieldPanel = new JPanel();
    yieldLabel = new JLabel("Yield");
    yieldField = new JTextField(4);
    yieldPanel.add(yieldLabel);
    yieldPanel.add(yieldField);

    JPanel densityPanel = new JPanel();
    densityLabel = new JLabel("Density");
    densityField = new JTextField(4);
    densityPanel.add(densityLabel);
    densityPanel.add(densityField);

    JPanel elasticModPanel = new JPanel();
    elasticModLabel = new JLabel("Elasitic Mod");
    elasticModField = new JTextField(4);
    elasticModPanel.add(elasticModLabel);
    elasticModPanel.add(elasticModField);

    add(yieldPanel);
    add(densityPanel);
    add(elasticModPanel);
  }

  /**Parse the input typed by the user. Return true if the input was
```

```
     *    successfully parsed.
     */
    public boolean parseInput(){
      if (!super.parseInput())
      return false;
      try{
      yield = Float.parseFloat(yieldField.getText());
      if (yield <= 0)
        throw new NumberFormatException();
      }
      catch(NumberFormatException e){
      JOptionPane.showMessageDialog(this,
                                "Yield is not valid.",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
      return false;
      }

      try{
      density = Float.parseFloat(densityField.getText());
      if (density <= 1000)
        throw new NumberFormatException();
      }
      catch(NumberFormatException e){
      JOptionPane.showMessageDialog(this,
                                "The density is not valid.",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
      return false;
      }

      try{
      elasticMod = Float.parseFloat(elasticModField.getText());
      if (elasticMod < 0)
        throw new NumberFormatException();
      }
      catch(NumberFormatException e){
      JOptionPane.showMessageDialog(this,
                                "The elastic mod is not valid.",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
      return false;
      }


      try{
      myMaterial = new Timber(id,yield,density,elasticMod);
      }
      catch(InvalidParameterException e){
      System.err.println(e.getMessage());
      System.exit(-1);
      }

      return true;
    }
}
```

```
  //Create the control pane for the top of the frame.
  private JPanel createControlPanel() {
    materialStrings = new Vector();
    materialStrings.add(" Metal ");
    materialStrings.add(" Timber ");
    boolean metalMaterialDefined = false;
    Vector materials = Structure.getInstance().getMaterials();
    for (int i = 0; i < materials.size(); i++){
      if (materials.elementAt(i) instanceof Metal){
      metalMaterialDefined = true;
      break;
      }
    }
    if (metalMaterialDefined){
      materialStrings.add(" ACI Concrete ");
      materialStrings.add(" B3 Concrete ");
    }
    final JComboBox layerList = new JComboBox(materialStrings);
    layerList.setSelectedIndex(0);
    layerList.addActionListener(new ActionListener () {
      public void actionPerformed(ActionEvent e) {
        int index = layerList.getSelectedIndex();
        JPanel oldPanel = currentPanel;
        if (index == 0)
          currentPanel = myMetalPanel;
        else if (index == 1)
          currentPanel = myTimberPanel;
        else if (index == 2)
          currentPanel = myACIConcretePanel;
        else if (index == 3)
          currentPanel = myB3ConcretePanel;
        if (oldPanel != currentPanel){
          oldPanel.setVisible(false);
          myElementContainerPanel.add(currentPanel);
          currentPanel.setVisible(true);
          NewMaterialDialog.this.setSize(500,500);
        }
      }
      });

    JPanel controls = new JPanel();
    controls.add(layerList);
    controls.setBorder(BorderFactory.createTitledBorder("Choose the
material."));
    return controls;
  }


  /**
   * The cancel/finish pane. Displays the cancel and finish
   * buttons.
   */
  private class CancelFinishPane extends JPanel{
    /** Finish Button */
    JButton myFinishBtn = null;
    /** Cancel Button */
```

```
    JButton myCancelBtn = null;

    /**Constructor*/
    public CancelFinishPane(){
      // create the buttons and register the actions
      myCancelBtn = new JButton("Cancel");
      myCancelBtn.addActionListener(myCancelAction);
      myCancelBtn.setToolTipText("Click to cancel.");
      myFinishBtn = new JButton("Finish");
      myFinishBtn.addActionListener(myFinishAction);
      myFinishBtn.setToolTipText("Click to finish.");

      //add the buttons to the pane
      BoxLayout layout = new BoxLayout(this, BoxLayout.X_AXIS);
      setLayout(layout);
      setBorder(myPaneEdgeBdr);
      add(Box.createHorizontalGlue());
      add(myCancelBtn);
      add(Box.createRigidArea(new Dimension(10, 0)));
      add(myFinishBtn);
    }
  }
  /**
   * Inner class that defines the cancel action
   */
  private class CancelAction extends AbstractAction {
    public void actionPerformed(ActionEvent evt) {
      setVisible(false);
    }
  }

  /**
   * Inner class that defines the finish action
   */
  private class FinishAction extends AbstractAction {
    public void actionPerformed(ActionEvent evt) {
      if (currentPanel.parseInput()){
      Structure.getInstance().addMaterial(myMaterial);
      setVisible(false);
      }
    }
  }
}
```

```
/*
 * @(#) NewSectionDialog.java
 *
 * History:   Mar 2002: Written by M.Gardner.
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

import javax.swing.*;
import javax.swing.border.*;
import javax.accessibility.*;

import java.util.*;
import java.awt.*;
import java.awt.event.*;

import structuralAnalyser.structuralElement.*;
import structuralAnalyser.material.*;

public class NewSectionDialog extends JDialog {
  /**The section created by the user */
  private Section mySection = null;

  /**The id for the section */
  protected String id = null;
  private JLabel idLabel = null;
  private JTextField idField = null;

  /**The area for the section */
  protected float area;
  private JLabel areaLabel = null;
  private JTextField areaField = null;

  /**The perimeter for the section */
  protected float perimeter;
  private JLabel perimeterLabel = null;
  private JTextField perimeterField = null;

  /**The imajor for the section */
  protected float imajor;
  private JLabel imajorLabel = null;
  private JTextField imajorField = null;

  /**The iminor for the section */
  protected float iminor;
  private JLabel iminorLabel = null;
  private JTextField iminorField = null;

  /** The pane that displays the cancel and finish buttons*/
  JPanel myCancelFinishPane = null;
  /** Finish Action */
  Action myFinishAction = new FinishAction();
  /** Cancel Action */
  Action myCancelAction = new CancelAction();
```

```
/**A border that puts 10 extra pixels around a pane.*/
Border myPaneEdgeBdr = BorderFactory.createEmptyBorder(10,10,10,10);

/**Constructor */
public NewSectionDialog(AnalyserFrame parent)
{
  super(parent,"New Section Editor",true);
  getContentPane().setLayout(new GridLayout(0,1));

  JPanel idPanel = new JPanel();
  idLabel = new JLabel("Name");
  idField = new JTextField(10);
  idPanel.add(idLabel);
  idPanel.add(idField);

  JPanel areaPanel = new JPanel();
  areaLabel = new JLabel("Area");
  areaField = new JTextField(4);
  //areaField.setText("100");
  areaPanel.add(areaLabel);
  areaPanel.add(areaField);

  JPanel perimeterPanel = new JPanel();
  perimeterLabel = new JLabel("Perimeter");
  perimeterField = new JTextField(4);
  perimeterPanel.add(perimeterLabel);
  perimeterPanel.add(perimeterField);

  JPanel imajorPanel = new JPanel();
  imajorLabel = new JLabel("I Major");
  imajorField = new JTextField(4);
  imajorPanel.add(imajorLabel);
  imajorPanel.add(imajorField);

  JPanel iminorPanel = new JPanel();
  iminorLabel = new JLabel("I Minor");
  iminorField = new JTextField(4);
  iminorPanel.add(iminorLabel);
  iminorPanel.add(iminorField);

  myCancelFinishPane = new CancelFinishPane();

  getContentPane().add(idPanel);
  getContentPane().add(areaPanel);
  getContentPane().add(perimeterPanel);
  getContentPane().add(imajorPanel);
  getContentPane().add(iminorPanel);
  getContentPane().add(myCancelFinishPane);
}

/**Parse the input typed by the user. Return true if the input was
 *   successfully parsed.
 */
public boolean parseInput(){
  id = idField.getText().trim();
  if (id.equals("") || id == null){
```

```
      JOptionPane.showMessageDialog(NewSectionDialog.this,
                                    "You must enter a name for the material",
                                    "Error",
                                    JOptionPane.ERROR_MESSAGE);
      id = null;
      return false;
    }
    //check that the id is unqiue
    Vector materials = Structure.getInstance().getMaterials();
    for (int i = 0; i < materials.size(); i++){
      Material material = (Material) materials.elementAt(i);
      if (material.getId().equals(id)){
      JOptionPane.showMessageDialog(NewSectionDialog.this,
                                    "This name has already been used for a material
in this structure.",
                                    "Error",
                                    JOptionPane.ERROR_MESSAGE);
      id = null;
      return false;
      }
    }

    try{
      area = Float.parseFloat(areaField.getText());
      if (area <= 0)
      throw new NumberFormatException();
    }
    catch (NumberFormatException e){
      JOptionPane.showMessageDialog(this,
                                    "The area of the section is not valid",
                                    "Error",
                                    JOptionPane.ERROR_MESSAGE);
      return false;
    }

    try{
      perimeter = Float.parseFloat(perimeterField.getText());
      if (perimeter <= 0)
      throw new NumberFormatException();
    }
    catch (NumberFormatException e){
      JOptionPane.showMessageDialog(this,
                                    "The perimeter of the section is not valid",
                                    "Error",
                                    JOptionPane.ERROR_MESSAGE);
      return false;
    }

    try{
      imajor = Float.parseFloat(imajorField.getText());
      if (imajor <= 0)
      throw new NumberFormatException();
    }
    catch (NumberFormatException e){
      JOptionPane.showMessageDialog(this,
                                    "The I-Major of the section is not valid",
                                    "Error",
```

```
                                JOptionPane.ERROR_MESSAGE);
      return false;
    }

    try{
      iminor = Float.parseFloat(iminorField.getText());
      if (iminor <= 0)
      throw new NumberFormatException();
    }
    catch (NumberFormatException e){
      JOptionPane.showMessageDialog(this,
                                "The I-Minor of the section is not valid",
                                "Error",
                                JOptionPane.ERROR_MESSAGE);
      return false;
    }

    try{
      mySection = new Section(id,area,perimeter,imajor,iminor);
    }
    catch(InvalidParameterException e){
      System.err.println(e.getMessage());
      System.exit(-1);
    }
    return true;
  }

  /**
   * The cancel/finish pane. Displays the cancel and finish
   * buttons.
   */
  private class CancelFinishPane extends JPanel{
    /** Finish Button */
    JButton myFinishBtn = null;
    /** Cancel Button */
    JButton myCancelBtn = null;

    /**Constructor*/
    public CancelFinishPane(){
      // create the buttons and register the actions
      myCancelBtn = new JButton("Cancel");
      myCancelBtn.addActionListener(myCancelAction);
      myCancelBtn.setToolTipText("Click to cancel.");
      myFinishBtn = new JButton("Finish");
      myFinishBtn.addActionListener(myFinishAction);
      myFinishBtn.setToolTipText("Click to finish.");

      //add the buttons to the pane
      BoxLayout layout = new BoxLayout(this, BoxLayout.X_AXIS);
      setLayout(layout);
      setBorder(myPaneEdgeBdr);
      add(Box.createHorizontalGlue());
      add(myCancelBtn);
      add(Box.createRigidArea(new Dimension(10, 0)));
      add(myFinishBtn);
    }
  }
```

```
  /**
   * Inner class that defines the cancel action
   */
  private class CancelAction extends AbstractAction {
    public void actionPerformed(ActionEvent evt) {
      setVisible(false);
    }
  }


  /**
   * Inner class that defines the finish action
   */
  private class FinishAction extends AbstractAction {
    public void actionPerformed(ActionEvent evt) {
      if (parseInput()){
      Structure.getInstance().addSection(mySection);
      setVisible(false);
      }
    }
  }

}
```

```
/*
 * @(#)NonFramingAlgorithm.java
 *
 * History:   Mar 2002: Written by  M. Gardner.
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

import structuralAnalyser.structuralElement.*;
import structuralAnalyser.material.*;

import java.util.*;
import java.io.*;
import java.lang.Math.*;

/**
 * This class contains the algorithm for calculating the shrinkage of
 * columns when framing is not taken into consideration. This class is
 * a singleton.
 */
public class NonFramingAlgorithm extends ShrinkageAlgorithm{

    private static NonFramingAlgorithm myInstance = null;

    public static NonFramingAlgorithm getInstance(){
      if (myInstance ==  null)
          myInstance = new NonFramingAlgorithm();
      return myInstance;
    }


     /**Calc the deltas for the columns by Framing over the period
(startDay,endDay) for existing loads
      * @param columns the columns to calculate the deltas for
     * @param startDay the start day of the calculations
     * @param endDay the end day of the calculations
     * @param internalLoads the internal loads (calculated from PI).
     * @return vector of columns with tempDelta set as the actual delta at the
endDay for existing loads
     */
    public Vector calcColumnDeltaExist(Vector columns, int startDay, int
endDay,
                           Vector internalLoads)
      throws InvalidParameterException
    {
      // need to find all creep,shrinkage and relaxation values. Creep from
startDay to EndDay
      //shrink and relax from 0 to EndDay subtract 0 to StartDay


  //System.out.println("calcColumnDeltaExist method");
      Vector columnsToReturn = new Vector();
      for(int n = 0; n<columns.size(); n++){
      columnsToReturn.addElement((Column)columns.elementAt(n));
```

```
      }
      //System.out.println(columnsToReturn + " columnsToReturn vector in
calcColumnDeltaExist method");
      // System.out.println(internalLoads + "internalLoads ");
      Vector newColumnList = new Vector();

      for(int i = 0; i<columns.size(); i++){
        Column aColumn = (Column)columns.elementAt(i);
        //System.out.println(aColumn + " aColumn  ");
        Load aLoad = (Load)internalLoads.elementAt(i);
        //System.out.println(aLoad + " aLoad  ");
        double aDelta = 0;
        //System.out.println(aColumn.getTempCreepPrevious() + " previousCreep
aColumn  ");
        //System.out.println(aColumn.getTempCreep() + " tempCreep Column  ");

//debugging. MG 27/9/03

        Float tempFloat = new Float(aLoad.getWeight());
        if (tempFloat.isNaN()){
          System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
          System.out.println(" Float is NaN " + tempFloat);
          System.out.println("At iteration " + i);
          //System.out.println("Loads are " + internalLoads);
          System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
          aLoad.setWeight(0);
        }
        //end of debugging

        if (aColumn.getConstructed())
          aDelta = makeCalculationExist(aColumn,aLoad, startDay, endDay);

        //   System.out.println(aDelta + " aDelta calculatd in
calcColumnDeltaExist method");

          // deltas.addElement(Double.toString(aDelta));
          aColumn.setTempDeltaExist(aDelta);


        newColumnList.addElement(aColumn);
      }


      Vector sortedStacks =
Structure.getInstance().getSortedStack(newColumnList);
      //System.out.println(sortedStacks + " sortedStacks in calcDeltaExist");
      //Vector sortedDeltaStacks = new Vector();
      for(int j = 0; j<sortedStacks.size(); j++){
        Vector aStack = (Vector)sortedStacks.elementAt(j);
//System.out.println(aStack);
        Vector sortedDeltasColumns = new Vector();
        double deltaOld = 0;

        for(int k = 0; k<aStack.size(); k++){
          Column colUnder = (Column)aStack.elementAt(k);
```

```
          double deltaOldTemp = colUnder.getTempDeltaExist();
          deltaOld = deltaOld + deltaOldTemp;
          colUnder.setTempDeltaExist(deltaOld);
          // System.out.println(deltaOld + " deltaOld in calcDeltaExist for
each column in stack");
          // sortedDeltasColumns.addElement(colUnder);
          //columnDeltas.addElement(colUnder);

        }  //sortedDeltaStacks.addElement(sortedDeltasColumns);

      }
      //Note columns are not in the same order as the came in but the
tempColumnDelta is set
      //so we return the copied vector pointing to the columns in the original
order.



      return  columnsToReturn;

    }



 /**Calc the deltas for the columns by Framing over the period
(startDay,endDay) for new loads
     * @param columns the columns to calculate the deltas for
     * @param startDay the start day of the calculations
     * @param endDay the end day of the calculations
     * @param internalLoads the internal loads (calculated from PI).
     * @return vector of columns with tempDeltaNew set as the actual delta at
the endDay for new loads
     */
    public Vector calcColumnDeltaNew(Vector columns, int startDay, int endDay,
                          Vector internalLoads)
      throws InvalidParameterException
    {
      // need to find elastic delta based on actual new internal loads at
endDay. If internal Load = 0 return 0.

      //System.out.println("calcColumnDeltaNew");
        // System.out.println(columns);
        //System.out.println(internalLoads);

      // System.out.println("calcColumnDeltaNew method");

      Vector columnsToReturn = new Vector();
      for(int i = 0; i<columns.size(); i++){
      columnsToReturn.addElement((Column)columns.elementAt(i));


      }
      //System.out.println(columnsToReturn + " columnsToReturn vector in
calcColumnDeltaNew method");
      Vector newColumnList = new Vector();
```

```
      for(int i = 0; i<columns.size(); i++){
        //System.out.println("i " +i);
        Column aColumn = (Column)columns.elementAt(i);
        Load aLoad = (Load)internalLoads.elementAt(i);
        double aDelta = 0;

        //debugging. GAL 26/9/03
        Float tempFloat = new Float(aLoad.getWeight());
        if (tempFloat.isNaN()){
          System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
          System.out.println(" Float is NaN " + tempFloat);
          System.out.println("At iteration " + i);
          //System.out.println("Loads are " + internalLoads);
          System.out.println("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!");
          aLoad.setWeight(0);
        }
        //end of debugging


        if(aLoad.getWeight()<=0){
          aColumn.setTempDeltaNew(0);
          //  System.out.println("a Column with no loads " +aColumn);
        }
        else if (aColumn.getConstructed()){
          //   System.out.println("a Column with  loads and constructed " +
aColumn);
          aDelta = makeCalculationNew(aColumn,aLoad, endDay);
          //System.out.println(aDelta + "aDelta in calcColumnDeltaNew method
for existing column with load");

          // deltas.addElement(Double.toString(aDelta));
          aColumn.setTempDeltaNew(aDelta);
        }

        newColumnList.addElement(aColumn);
      }

      Vector sortedStacks =
Structure.getInstance().getSortedStack(newColumnList);
      //System.out.println(sortedStacks + " sortedStacks in calcDeltaNew");
      //System.out.println(sortedStacks);
      //Vector sortedDeltaStacks = new Vector();
      for(int j = 0; j<sortedStacks.size(); j++){
        Vector aStack = (Vector)sortedStacks.elementAt(j);
//System.out.println(aStack);
        //Vector sortedDeltasColumns = new Vector();
        double deltaOld = 0;

        for(int k = 0; k<aStack.size(); k++){
          Column colUnder = (Column)aStack.elementAt(k);
          double deltaOldTemp = colUnder.getTempDeltaNew();
          deltaOld = deltaOld + deltaOldTemp;
          colUnder.setTempDeltaNew(deltaOld);
          //   System.out.println(deltaOld + " deltaOld in calcDeltaNew for
each column in stack");
          //sortedDeltasColumns.addElement(colUnder);
```

```
          //columnDeltas.addElement(colUnder);

        }  //sortedDeltaStacks.addElement(sortedDeltasColumns);

      }
      //Note columns are not in the same order as the came in but the
tempColumnDeltaNew is set
      //so we return the copied vector pointing to the columns in the original
order.



      return  columnsToReturn;

    }



    /**Calc the deltas for the columns by Framing over the period
(startDay,endDay)
     * @param columns the columns to calculate the deltas for
     * @param startDay the start day of the calculations
     * @param endDay the end day of the calculations
 * @return a vector of deltas for each column
     */

    public Vector calcColumnDelta(Vector columns, int startDay, int endDay)
      throws InvalidParameterException
    {
      return calcColumnDelta(columns,endDay); // for now
    }



  /**Calc the deltas values for a group of column by NonFraming
   * @param columns the vector of columns
   * @param day the day of calculation
   * @return a vector of columns with tempDelta set.
   */
    public Vector calcColumnDelta(Vector columns, int day)
      throws InvalidParameterException
    {
      // System.out.println("this method for calc Column Deltas");
      //  System.out.println(columns);
      //  System.out.println(day);
      Vector columnDeltas = new Vector();
      //Vector deltas = new Vector();
      Vector forces = calcColumnForces(columns,0,day);
      //System.out.println(forces+"  here force");
      Vector newColumnList = new Vector();
```

```
      for(int i = 0; i<columns.size(); i++){
        Column aColumn = (Column)columns.elementAt(i);
        //System.out.println(aColumn+"  here ");
        Vector columnForces = (Vector)forces.elementAt(i);
        double aDelta = makeCalculation(aColumn,columnForces, day);
        //  System.out.println(aDelta);

        // deltas.addElement(Double.toString(aDelta));
        aColumn.setTempDelta(aDelta);
        newColumnList.addElement(aColumn);
      }

      Vector sortedStacks =
Structure.getInstance().getSortedStack(newColumnList);
      //System.out.println(sortedStacks);
      Vector sortedDeltaStacks = new Vector();
      for(int j = 0; j<sortedStacks.size(); j++){
        Vector aStack = (Vector)sortedStacks.elementAt(j);
        //System.out.println(aStack);
        Vector sortedDeltasColumns = new Vector();
        double deltaOld = 0;

        for(int k = 0; k<aStack.size(); k++){
          Column colUnder = (Column)aStack.elementAt(k);
          double deltaOldTemp = colUnder.getTempDelta();
          deltaOld = deltaOld + deltaOldTemp;
          colUnder.setTempDelta(deltaOld);
          sortedDeltasColumns.addElement(colUnder);
          columnDeltas.addElement(colUnder);

        }
        sortedDeltaStacks.addElement(sortedDeltasColumns);
      }
      //Note columns are not in the same order as the came in but the
tempColumnDelta is set



      //System.out.println(columnDeltas +"these are the column deltas");

      return  columnDeltas;
    }



 /**Calc the total force  values in each column for a group of column by
NonFraming
   * @param columns the vector of columns
   * @param startDay the day of calculation
   * @param endDay the day of calculation
   * @return a vector of forces for each column
   */
  public Vector calcColumnForces(Vector columns, int startDay, int endDay)
  throws InvalidParameterException
  {
    Vector forces = new Vector();
```

```
        for(int i = 0; i<columns.size(); i++){
            Column thisColumn = (Column) columns.elementAt(i);
          //  System.out.println(thisColumn+" i = "+i);
          /*The stack of columns above the given column*/
          Vector stackedColumns =
Structure.getInstance().getColumnStack(thisColumn,endDay);

          //    System.out.println(stackedColumns);
          Vector columnForces = new Vector();

          /*The vector of Loads for the first column in stack*/
          Vector thisColumnLoads =
Structure.getInstance().getAllLoads(thisColumn);
          //    System.out.println(thisColumnLoads + " all the loads for this
column ");
               for(int k = 1; k<thisColumnLoads.size(); k++){
               Load aLoad = (Load)thisColumnLoads.elementAt(k);
               //   System.out.println(aLoad+ " k = "+k);
               if (aLoad.getDayConstructed() < endDay)
                 columnForces.addElement(aLoad);
               //   System.out.println(columnForces+ " first column forces ");
             }


            for(int j = 1; j<stackedColumns.size(); j++){
            Column theNextColumn = (Column) stackedColumns.elementAt(j);
            /*The vector of loads for the next columns in the stack*/
            Vector thisNextColumnLoads =
Structure.getInstance().getAllLoads(theNextColumn);
                 for(int l = 0; l<thisNextColumnLoads.size(); l++){
                 Load aNextLoad = (Load)thisNextColumnLoads.elementAt(l);
                 //    System.out.println(aNextLoad + " j = "+j+" l = "+l );
                 if (aNextLoad.getDayConstructed() < endDay)
                   columnForces.addElement(aNextLoad);
                 // System.out.println(columnForces+ " next column forces ");
               }
          }
            //   System.out.println(forces+ " here we are ");
          forces.addElement(columnForces);
          // System.out.println(forces+ " here we are ");
      }




    return forces;

  }


  //  public double calcColumnDelta(Column column, int day)
  //throws InvalidParameterException{
```

```
  //if (day < column.getDayConstructed())
  //    throw new  InvalidParameterException("The day is before the column was
added.");
  //if (column == null)
  //  throw new  InvalidParameterException("The column is null.");

      /*The stack of columns above the given column*/
  //Vector columns = Structure.getInstance().getColumnStack(column,day);
      //System.out.println("columns " + columns);
      /**The shrinkage*/
  //double delta = 0;
      /**The vector of days for which a force acts on the discrete column*/
  //Vector days = new Vector();
      /**The vector of forces acting on the column*/
  //Vector forces = new Vector();
  //float force = 0;

      //for each day, for the selected column, make the shrinkage
      //calculation at each day a force is added. The delta
      //calculated is the discrete axial shortening value for the
      //column.
  //for (int d = column.getDayConstructed()+1; d<= day; d++){
  //    force = 0;
  //    for (int c = 0; c < columns.size(); c++){
            //examine a column from the stack of columns
            //System.out.println("in here " + c);
  //  Column stackedColumn = (Column) columns.elementAt(c);
            //get all the forces on the stacked column for a given day
  //  force = force +  Structure.getInstance().getForce(stackedColumn,d);
            //System.out.println("force " + force);
  //    }
  //    if(force > 0){
  //  days.addElement(new Integer(d));
  //  forces.addElement(new Float(force));
  //    }
  //}

      //System.out.println("forces " + forces);
  //delta = makeCalculation(column,days,forces,day);
  //return delta;
  //}


 /**
     * Make an actual calculation of a particular column delta for a
particular
     * set of forces and time
     * @param column the actual column in question
     * @param loads the total applied forces on the column up to the day
     * @param day the day for which we want to calculate delta.
     * @return a double value for the total delta of the column due all the
forces up to the day required
     */
    private double makeCalculation(Column column,
                           Vector loads, int day)
    {
```

```
      //System.out.println(column);
      Section thisSection = column.getSection();

      float myArea = thisSection.getArea();

      double creepmultiplier = 1.088;

      if (myArea < 10){
        creepmultiplier = 1.62;

        if (myArea > 1.71)
          creepmultiplier = 1.19;
      }

//  System.out.println(column +" " + loads +" " + day  +" in make
calculation");
int day0 = column.getDayConstructed();
      double totalE = 0;
      double totalC = 0;
      double totalS = 0;
      double totalR = 0;
      Vector thisDayMaterial = new Vector();
      Vector creepCoeff  = new Vector();
      double shrinkCoeff = calcShrinkCoeff(column, day0, day);
      // System.out.println(shrinkCoeff);
      for  (int i = 0; i < loads.size(); i++){
        double aTotalE = 0;
        double aTotalC = 0;
        Load aLoad = (Load)loads.elementAt(i);
        //  System.out.println(aLoad);
        int loadDay = aLoad.getDayConstructed();
        //  System.out.println(loadDay);
        float  aForce =
Float.valueOf(Double.toString(aLoad.getWeight())).floatValue();
        //   System.out.println(aForce+"aForce i = "+i);

            creepCoeff = calcCreepCoeff(column, day0, loadDay, day);
            // System.out.println(creepCoeff + " creepCoeff");
            thisDayMaterial = calcMaterial(column, creepCoeff, day0, loadDay,
day);
            // System.out.println(thisDayMaterial + "thisDayMaterial ");
            aTotalE = aTotalE + calcElastic(column, thisDayMaterial, aForce);
            // System.out.println(aTotalE + "aTotalE ");
            aTotalC = aTotalC -aTotalE + calcCreep(column, thisDayMaterial,
aForce);
            // System.out.println(aTotalC + "aTotalC ");
            //totalR = calcRelax(column, shrinkCoeff, thisDayMaterial);
            thisDayMaterial.removeAllElements();
            creepCoeff.removeAllElements();

            totalE = totalE + aTotalE;
            totalC = totalC + aTotalC;

          }
          totalC = totalC * creepmultiplier;
          creepCoeff = calcCreepCoeff(column, day0, day);
          // System.out.println(creepCoeff + " creepCoeff");
```

```
            thisDayMaterial = calcMaterial(column, creepCoeff, day0,  day);
            // System.out.println(thisDayMaterial + "thisDayMaterial ");
            totalS = calcShrink(column, shrinkCoeff);
            totalR = calcRelax(column, shrinkCoeff, thisDayMaterial);




            //  System.out.println("Here for non framing");
            // System.out.println(totalE);
            //System.out.println(totalC);
            //System.out.println(totalS);
            //System.out.println(totalR);
        double total = totalE + totalC + totalS + totalR;

        return total;
    }


/**
     * Make an actual calculation of a particular column delta for a
particular
     * set of forces and time
     * @param column the actual column in question
     * @param loads the total applied forces on the column up to the day
     * @param day the day for which we want to calculate delta.
     * @return a double value for the total delta of the column due all the
forces up to the day required
     */
    private double makeCalculationExist(Column column,
                            Load load, int startDay, int endDay)
    {
      // double test = 0.5;
      //return test;

      // the rest of this method has been commented out for testing purposes

      //System.out.println(column);
      Section thisSection = column.getSection();

      float myArea = thisSection.getArea();

      double creepmultiplier = 1.088;

      if (myArea < 10){
        creepmultiplier = 1.62;

        if (myArea > 1.71)
          creepmultiplier = 1.25;
      }

      // System.out.println(column);
        //System.out.println(loads);
        int day0 = column.getDayConstructed();
        double totalE = 0;     //elastic at start day
        double totalC = 0;
```

```
        double totalS = 0;
      double totalR = 0;
        double previousCreep = column.getTempCreepPrevious();
      //  System.out.println(previousCreep + " previousCreep  ");
      //  System.out.println(column.getTempCreep() + " tempCreep    ");
        double totalSStart = 0;
        double totalSEnd = 0;
        double totalRStart = 0;
        double totalREnd = 0;
        Vector thisDayMaterial = new Vector();
        Vector creepCoeff  = new Vector();
        double shrinkCoeffStart = calcShrinkCoeff(column, day0, startDay);
        double shrinkCoeffEnd = calcShrinkCoeff(column, day0, endDay);


        int loadDay = startDay + 1;
      Float tempFloat = new Float(load.getWeight());
        float aForce  = tempFloat.floatValue();
        //float  aForce =
Float.valueOf(Double.toString(load.getWeight())).floatValue();


        creepCoeff = calcCreepCoeff(column, day0, loadDay, endDay);
        thisDayMaterial = calcMaterial(column, creepCoeff, day0, loadDay,
endDay);

        totalE = calcElastic(column, thisDayMaterial, aForce);
        totalC = -totalE + calcCreep(column, thisDayMaterial, aForce);//Look
at this
      // System.out.println(totalC + "totalCExist before deduction");
        column.setTempCreep(totalC);

        totalC = totalC - previousCreep;
      totalC = totalC*creepmultiplier;
      // System.out.println(totalC + "totalCExist after deduction");
        thisDayMaterial.removeAllElements();
        creepCoeff.removeAllElements();


        creepCoeff = calcCreepCoeff(column, day0, startDay);
        thisDayMaterial = calcMaterial(column, creepCoeff, day0,  startDay);
        totalSStart = calcShrink(column, shrinkCoeffStart);
        totalRStart = calcRelax(column, shrinkCoeffStart, thisDayMaterial);

      thisDayMaterial.removeAllElements();
        creepCoeff.removeAllElements();


        creepCoeff = calcCreepCoeff(column, day0, endDay);
        thisDayMaterial = calcMaterial(column, creepCoeff, day0,  endDay);
        totalSEnd = calcShrink(column, shrinkCoeffEnd);
        totalREnd = calcRelax(column, shrinkCoeffEnd, thisDayMaterial);

        totalS = totalSEnd - totalSStart;
        totalR = totalREnd - totalRStart;

        //System.out.println(totalE + "totalEExist");
```

```
        //System.out.println(totalC + "totalCExist used");
        //System.out.println(totalS + "totalSExist");
        //System.out.println(totalR + "totalRExist");

        double total = totalC + totalS + totalR;

        return total;
    }


 /**
     * Make an actual calculation of a particular column delta for a
particular
     * set of forces and time
     * @param column the actual column in question
     * @param load the total applied force on the column up to the day
     * @param day the day for which we want to calculate delta.
     * @return a double value for the total elastic delta of the column due
all the forces up to the day required
     */
    private double makeCalculationNew(Column column,
                          Load load, int day)
    {

      // double test = 1;
      // return test;

      // the rest of this method has been commented out for testing purposes


        //System.out.println(loads);
        int day0 = column.getDayConstructed();
      double totalE = 0;
      Vector thisDayMaterial = new Vector();
      Vector creepCoeff  = new Vector();

        // System.out.println(aLoad);
        int loadDay = day;
        //System.out.println(loadDay);
        Float tempFloat = new Float(load.getWeight());
        float aForce  = tempFloat.floatValue();
        // System.out.println(aForce+"aForce i = "+i);

        creepCoeff = calcCreepCoeff(column, day0, loadDay, day);
        thisDayMaterial = calcMaterial(column, creepCoeff, day0, loadDay,
day);
        totalE = calcElastic(column, thisDayMaterial, aForce);

        //System.out.println(totalE + "totalENew");
        thisDayMaterial.removeAllElements();
        creepCoeff.removeAllElements();



      return totalE;
    }
```

```
 /**
     * Make an calculation of the creep coeff for the material of the entire
age of the column
     * @param column the actual column in queation
     * @param timeCreated the day the column was created

     * @param timeCalc the day for which we want to calculate delta.
     * @return a Vector of the coefficient of creep
     */
    private Vector calcCreepCoeff (Column column, int timeCreated,
                           int timeCalc){
      Vector creeps = new Vector();

      float mySlump = 0;
      float myPercentFine = 0;
      float myPercentAir = 0;
      float myHumidity =0;

      Material m = column.getMaterial();


      if (m instanceof ACIConcrete){
        ACIConcrete thisMaterial= (ACIConcrete)m;
        mySlump = thisMaterial.getSlump() ;
        myPercentFine = thisMaterial .getPercentFineAggregate();
        myPercentAir = thisMaterial .getPercentAir();
        myHumidity  = thisMaterial .getHumidity();
        //density = thisMaterial.getDensity();
      }


      Section thisSection = column.getSection();

      float myArea = thisSection.getArea() * 1000000;
      float myPerimeter = thisSection.getPerimeter() * 1000;
      float myRatio = 4 * myArea / myPerimeter;



      //int ageS = timeForce - timeCreated;
      //int ageTS = timeCalc - timeForce;
      int ageT0 = timeCalc - timeCreated;

      //double myCreepS = .01;
      //double myCreepTS = .01;
      double myCreepT0 = .01;


      double myMultTime =  1;
      double myMultHumid = 1;
      double myMultSurfRatio = 1;
      double myMultSlump = 0.82 + (0.00264 * mySlump);
      double myMultPercentFine =0.88 + (0.0024 * myPercentFine);
      //double myMultSlump = 1;
```

```
        //double myMultPercentFine =1;
        double myMultPercentAir = 1;



        //if (ageT0 > 7)
        // myMultTime = 1.25 * Math.pow(ageT0, -.118);


        if( myHumidity > 40)
         myMultHumid = 1.27 - (0.0067 * myHumidity);

        if (myRatio < 150)
          myMultSurfRatio = 1.30 - (0.003 * (myRatio-50));

        if (myRatio >150  && myRatio <= 380){
          if (ageT0 < 365)
            myMultSurfRatio = 1.14 - (0.00092 * myRatio);
          else
            myMultSurfRatio = 1.1 - (0.00067 * myRatio);
        }

        if (myRatio > 380)
           myMultSurfRatio =( 2 * (1.13 * Math.pow(Math.E,(-0.005325 * myRatio))
+ 1)) / 3;
        // Altered for Baoke from
        // myMultSurfRatio =( 2 * (1.13 * Math.pow(Math.E,(-0.002 * myRatio)) +
1)) / 3;

        if (0.46 + 0.09 * myPercentAir >= 1)
          myMultPercentAir = 0.46 + (0.09 * myPercentAir);



        //double ultimateCreep = 2.35 * myMultTime * myMultHumid *
myMultSurfRatio * myMultSlump * myMultPercentFine * myMultPercentAir;
        // Adopt creep base value of 1.15 for bayoke
        double ultimateCreep = 1.15 * myMultTime * myMultHumid * myMultSurfRatio
* myMultSlump * myMultPercentFine * myMultPercentAir;
        //System.out.println(myMultTime+"myMultTime");
        //System.out.println( myMultHumid+" myMultHumid");
        //System.out.println( myMultSurfRatio+" myMultSurfRatio");

        //System.out.println(myMultSlump+"myMultSlump");
        //System.out.println(myMultPercentFine+"myMultPercentFine");
        //System.out.println(myMultPercentAir+"myMultPercentAir");

        //System.out.println(ultimateCreep+"ultimateCreep");

        //if (ageS >0)
        //  myCreepS =  ultimateCreep * (Math.pow(ageS,0.6) / (10 +
Math.pow(ageS,0.6))) ;
        //if (ageTS >0)
        //  myCreepTS = ultimateCreep * (Math.pow(ageTS,0.6) / (10 +
Math.pow(ageTS,0.6)));
        if (ageT0 >0)
```

```
          myCreepT0 = ultimateCreep * (Math.pow(ageT0,0.6) / (10 +
Math.pow(ageT0,0.6)));
      //double myageAdjustmentS = 0.8 * myCreepS;
      //double myageAdjustmentTS = 0.8 * myCreepTS;
      double myageAdjustmentT0 = 0.8 * myCreepT0;
        //System.out.println(myCreepS+"myCreepS");
      //System.out.println(myCreepTS+"myCreepTS");
      //System.out.println(myCreepT0+"myCreepTO");


      //creeps.addElement(new Double(myCreepS));
      //creeps.addElement(new Double(myCreepTS));
      creeps.addElement(new Double(myCreepT0));
      //creeps.addElement(new Double(myageAdjustmentS));
      //creeps.addElement(new Double(myageAdjustmentTS));
      creeps.addElement(new Double(myageAdjustmentT0));
      //System.out.println(creeps);
      return creeps;
    }




    /**
     * Make an calculation of the creep coeff for a particular force and time
     * @param column the actual column in queation
     * @param timeCreated the day the column was created
     * @param timeForce the day of the applied force
     * @param timeCalc the day for which we want to calculate delta.
     * @return a Vector of the coefficient of creep
     */
    private Vector calcCreepCoeff (Column column, int timeCreated,
                            int timeForce, int timeCalc){
      Vector creeps = new Vector();

      float mySlump = 0;
      float myPercentFine = 0;
      float myPercentAir = 0;
      float myHumidity = 0;
      Section thisSection = column.getSection();
      Material m = column.getMaterial();


      if (m instanceof ACIConcrete){
        ACIConcrete thisMaterial= (ACIConcrete)m;
        mySlump = thisMaterial.getSlump() ;
        myPercentFine = thisMaterial .getPercentFineAggregate();
        myPercentAir = thisMaterial .getPercentAir();
        myHumidity  = thisMaterial .getHumidity();
        //density = thisMaterial.getDensity();
      }



      float myArea = thisSection.getArea() * 1000000;
      float myPerimeter = thisSection.getPerimeter() * 1000;
```

```
        float myRatio = 4 * myArea / myPerimeter;


        int ageS = timeForce - timeCreated;
        int ageTS = timeCalc - timeForce;
        int ageT0 = timeCalc - timeCreated;

        double myCreepS = .0000001;
        double myCreepTS = .0000001;
        double myCreepT0 = .0000001;

        double myMultTime =  1;
        double myMultHumid = 1;
        double myMultSurfRatio = 1;
        double myMultSlump = 0.82 + (0.00264 * mySlump);
        double myMultPercentFine =0.88 + (0.0024 * myPercentFine);
        //double myMultSlump = 1;
        //double myMultPercentFine =1;
        double myMultPercentAir = 1;



        if (ageS > 7)
         myMultTime = 1.25 * Math.pow(ageS, -.118);


        if( myHumidity > 40)
         myMultHumid = 1.27 - (0.0067 * myHumidity);

        if (myRatio < 150)
          myMultSurfRatio = 1.30 - (0.003 * (myRatio-50));

        if (myRatio >150  && myRatio <= 380){
          if (ageTS < 365)
            myMultSurfRatio = 1.14 - (0.00092 * myRatio);
          else
            myMultSurfRatio = 1.1 - (0.00067 * myRatio);
        }

        if (myRatio > 380)
           myMultSurfRatio =( 2 * (1.13 * Math.pow(Math.E,(-0.005325 * myRatio))
+ 1)) / 3;
        // Altered for Baoke from
        // myMultSurfRatio =( 2 * (1.13 * Math.pow(Math.E,(-0.002 * myRatio)) +
1)) / 3;
        if (0.46 + 0.09 * myPercentAir >= 1)
          myMultPercentAir = 0.46 + (0.09 * myPercentAir);


        //double ultimateCreep = 2.35 * myMultTime * myMultHumid *
myMultSurfRatio * myMultSlump * myMultPercentFine * myMultPercentAir;
        // Adopt creep base value of 1.15 for bayoke
        double ultimateCreep = 1.15 * myMultTime * myMultHumid * myMultSurfRatio
* myMultSlump * myMultPercentFine * myMultPercentAir;


        //System.out.println(myMultTime+"myMultTime");
```

```
        //System.out.println( myMultHumid+" myMultHumid");
        //System.out.println( myMultSurfRatio+" myMultSurfRatio");

        //System.out.println(myMultSlump+"myMultSlump");
        //System.out.println(myMultPercentFine+"myMultPercentFine");
        //System.out.println(myMultPercentAir+"myMultPercentAir");

        //System.out.println(ultimateCreep+"ultimateCreep");
        if (ageS >0)
            myCreepS =  ultimateCreep * (Math.pow(ageS,0.6) / (10 +
Math.pow(ageS,0.6))) ;
        if (ageTS >0)
            myCreepTS = ultimateCreep * (Math.pow(ageTS,0.6) / (10 +
Math.pow(ageTS,0.6)));
        if (ageT0 >0)
            myCreepT0 = (ultimateCreep * (Math.pow(ageT0,0.6) / (10 +
Math.pow(ageT0,0.6))))/myMultTime;
        double myageAdjustmentS = 0.8 * myCreepS;
        double myageAdjustmentTS = 0.8 * myCreepTS;
        double myageAdjustmentT0 = 0.8 * myCreepT0;
          //System.out.println(myCreepS+"myCreepS");
        //System.out.println(myCreepTS+"myCreepTS");
        //System.out.println(myCreepT0+"myCreepTO");


        creeps.addElement(new Double(myCreepS));
        creeps.addElement(new Double(myCreepTS));
        creeps.addElement(new Double(myCreepT0));
        creeps.addElement(new Double(myageAdjustmentS));
        creeps.addElement(new Double(myageAdjustmentTS));
        creeps.addElement(new Double(myageAdjustmentT0));
        //System.out.println(creeps);
        return creeps;
    }


    /**
     * Make an calculation of the shrinkage coeff for a particular force and
     time
     * @param column the actual column in queation
     * @param timeCreated the day the column was created
     * @param timeCalc the day for which we want to calculate delta.
     * @return a Vector of the coefficient of shrinkage
     */
    private double calcShrinkCoeff (Column column, int timeCreated,
                            int timeCalc){

        double shrinks = 0;

        Section thisSection = column.getSection();
        Material m = column.getMaterial();

        float mySlump = 0;
        float myPercentFine = 0;
        float myPercentAir = 0;
        float myCementContent = 0;
        float myHumidity = 0;
```

```
float myCuringtime = 0;
double myBasicShrink =0;
if (m instanceof ACIConcrete){
  ACIConcrete thisMaterial= (ACIConcrete)m;
  mySlump = thisMaterial.getSlump() ;
  myPercentFine = thisMaterial .getPercentFineAggregate();
  myPercentAir = thisMaterial .getPercentAir();
  myCementContent = thisMaterial.getCementContent();
  myHumidity  = thisMaterial .getHumidity();
  myCuringtime  = thisMaterial .getCuringTime();
  myBasicShrink = thisMaterial.getBasicShrinkage();

}


int ageTd = timeCalc - timeCreated - 2;
float myArea = thisSection.getArea() * 1000000;
float myPerimeter = thisSection.getPerimeter() * 1000;
float myRatio = 4 * myArea / myPerimeter;
//System.out.println(myRatio + "ratio");




//double myBasicShrink = .00078;
double myMultTime =  1;// this is a default value for 7 days wet.
double myMultHumid = 1;
double myMultSurfRatio = 1;
double myMultSlump = 0.89 + (0.00161 * mySlump);
//double myMultSlump =1;
double myMultPercentFine = 1;
double myMultCementCont =0.75 + (0.00061 * myCementContent);
double myMultPercentAir = 0.95 + (0.008 * myPercentAir);
//double myMultCementCont =1.0245;
//double myMultPercentAir =1.1;

if(myCuringtime < 7){
  if(myCuringtime < 3)
    myMultTime = 1.2;
else  myMultTime = 1.1;
}

if(myCuringtime > 7){
  if(myCuringtime > 14)
     if(myCuringtime > 28)
       myMultTime = 0.75;
     else  myMultTime = 0.86;
  else  myMultTime = .93;
}

//System.out.println(myMultTime + "myMultTime");

if(myHumidity > 80)
    myMultHumid = 3 - (0.03 * myHumidity);
```

```
        if(myHumidity >= 40)
            myMultHumid = 1.4 -( 0.01 * myHumidity);

        if(myHumidity > 80)
            myMultHumid = 3 - (0.03 * myHumidity);


        if (myRatio < 150)
          myMultSurfRatio = 1.35 - (0.0035 * (myRatio-50));

        else if (myRatio < 380 && ageTd > 365)
          myMultSurfRatio = 1.17 -( 0.00114 * myRatio);

        else if (myRatio < 380 && ageTd > 365)
          myMultSurfRatio = 1.23 - (0.0015 * myRatio);

        else
            myMultSurfRatio = 1.2 * Math.pow(Math.E,(-0.00118 * myRatio));

        if ( myMultSurfRatio < 0.2)
          myMultSurfRatio = 0.2;

        if (myPercentFine <= 50)
          myMultPercentFine = 0.3 + (0.014 * myPercentFine);
          else
          myMultPercentFine = 0.9 + (0.002 * myPercentFine);

        //double ultimateShrink = myBasicShrink  * myMultTime * myMultHumid *
        //   myMultSurfRatio * myMultSlump * myMultPercentFine *
        //   myMultCementCont * myMultPercentAir;

        // Basic shrink div 0.62 for Bayoke

        double ultimateShrink = myBasicShrink  * myMultTime * myMultHumid *
            myMultSurfRatio * myMultSlump * myMultPercentFine *
            myMultCementCont * myMultPercentAir / 0.62;

        //System.out.println(myBasicShrink + " myBasicShrink ");
        //System.out.println(myMultTime + "  myMultTime");
        //System.out.println( myMultHumid + "  myMultHumid ");
        //System.out.println(myMultSurfRatio + " myMultSurfRatio ");
        //System.out.println( myMultSlump + "  myMultSlump ");
        //System.out.println(myMultPercentFine + " myMultPercentFine ");
        //System.out.println(myMultCementCont + " myMultCementCont ");
        //System.out.println(myMultPercentAir + " myMultPercentAir ");

        if (ageTd > 0)
            shrinks  =  ultimateShrink * ageTd  / (35 + ageTd) ;
        //System.out.println(shrinks+"shirink coeff");

        return shrinks;
    }


/**
     * Make an calculation of the material properties for the entire time
     * @param conc the actual concrete material in the column
```

```
     * @param creepCoeff the creep coefficients of the material on the
     * day in question
     * @param timeCreated the day the column was created

     * @param timeCalc the day for which we want to calculate delta.
     * @return a Vector of the material properties at the time of
     * calculation
     */
    private Vector calcMaterial (Column column, Vector creepCoef,
                         int timeCreated,
                         int timeCalc)
  {

      Material m = column.getMaterial();
      Metal thisMetal = null;
      float myfc28 = 0;
      float myDensity = 0;
      String myCementType = null;

      if (m instanceof ACIConcrete){
        ACIConcrete thisMaterial= (ACIConcrete)m;
        thisMetal = thisMaterial.getAddedMetal();
        myfc28 = thisMaterial.getConc28day();
        myDensity = thisMaterial.getDensity();
        myCementType = thisMaterial.getCementType();
      }

      Vector  myMaterialsCalculated = new Vector();
      //double myCreepS
=Double.valueOf(creepCoef.elementAt(0).toString()).doubleValue();
      //double myCreepTS
=Double.valueOf(creepCoef.elementAt(1).toString()).doubleValue();
      double myCreepT0
=Double.valueOf(creepCoef.elementAt(0).toString()).doubleValue();
      //double myageAdjustmentS
=Double.valueOf(creepCoef.elementAt(3).toString()).doubleValue();
      //double myageAdjustmentTS
=Double.valueOf(creepCoef.elementAt(4).toString()).doubleValue();
      double myageAdjustmentT0 =
Double.valueOf(creepCoef.elementAt(1).toString()).doubleValue();
      //System.out.println(myageAdjustmentT0 + "myageAdjustmentT0 in
calcMaterial shrinkR");

      float myESteel = thisMetal.getElasticMod();

      //int ageS = timeForce - timeCreated;
      //int ageTS = timeCalc - timeForce;

      double newDensityFactor = 0.043 * Math.pow(myDensity,1.5); //raise
m_density to 1.5
      double fc0 = calcConcStrengthFactor(myCementType,2) * myfc28;
      //System.out.println(fc0 + "fc0 in calcMaterial shrinkR");
      //double fcS = calcConcStrengthFactor(myCementType,ageS) * myfc28;

      double Ec0 = newDensityFactor * Math.sqrt(fc0); // square root of fc0
      //double EcS = newDensityFactor * Math.sqrt(fcS); // square root of fcS
      //System.out.println(Ec0 + "Ec0 in calcMaterial shrinkR");
```

```
      //double EeTS = EcS / (1 + myCreepTS) ;
      //double EeAdjustTS = EcS / (1 + myageAdjustmentTS
      double EeAdjustT0 = Ec0 / (1 + myageAdjustmentT0) ;
      //System.out.println(EeAdjustT0 + "EeAdjustT0 in calcMaterial shrinkR");
      //double nS = myESteel / EcS;
      //double neTS = myESteel / EeTS;
      //double neAdjustTS = myESteel / EeAdjustTS;
      double neAdjustT0 = myESteel / EeAdjustT0;
      //System.out.println(neAdjustT0 + "neAdjustT0  in calcMaterial
shrinkR");
      //myMaterialsCalculated.addElement(new Double(EcS));
      //myMaterialsCalculated.addElement(new Double(nS));
      //myMaterialsCalculated.addElement(new Double(EeTS));
      //myMaterialsCalculated.addElement(new Double(neTS));
      //myMaterialsCalculated.addElement(new Double(EeAdjustTS));
      //myMaterialsCalculated.addElement(new Double(neAdjustTS));
      //myMaterialsCalculated.addElement(new Double(EeAdjustT0));
      myMaterialsCalculated.addElement(new Double(neAdjustT0));

      return myMaterialsCalculated;
    }




    /**
     * Make an calculation of the material properties for a particular
     * force and time
     * @param conc the actual concrete material in the column
     * @param creepCoeff the creep coefficients of the material on the
     * day in question
     * @param timeCreated the day the column was created
     * @param timeForce the day of the applied force
     * @param timeCalc the day for which we want to calculate delta.
     * @return a Vector of the material properties at the time of
     * calculation
     */
    private Vector calcMaterial (Column column, Vector creepCoef,
                          int timeCreated,  int timeForce,
                          int timeCalc)
  {

      Material m = column.getMaterial();
      Metal thisMetal = null;
      float myfc28 = 0;
      float myDensity = 0;
      String myCementType =null;

      if (m instanceof ACIConcrete){
        ACIConcrete thisMaterial= (ACIConcrete)m;

        thisMetal = thisMaterial.getAddedMetal();
        myfc28 = thisMaterial.getConc28day();
        myDensity = thisMaterial.getDensity();
        myCementType =  thisMaterial.getCementType();
      }
```

```
        float myESteel = thisMetal.getElasticMod();

        Vector  myMaterialsCalculated = new Vector();
        double myCreepS
=Double.valueOf(creepCoef.elementAt(0).toString()).doubleValue();
        double myCreepTS
=Double.valueOf(creepCoef.elementAt(1).toString()).doubleValue();
        double myCreepT0
=Double.valueOf(creepCoef.elementAt(2).toString()).doubleValue();
        double myageAdjustmentS
=Double.valueOf(creepCoef.elementAt(3).toString()).doubleValue();
        double myageAdjustmentTS
=Double.valueOf(creepCoef.elementAt(4).toString()).doubleValue();
        double myageAdjustmentT0 =
Double.valueOf(creepCoef.elementAt(5).toString()).doubleValue();



        int ageS = timeForce - timeCreated;
        int ageTS = timeCalc - timeForce;

        double newDensityFactor = 0.043 * Math.pow(myDensity,1.5); //raise
m_density to 1.5
        double fc0 = calcConcStrengthFactor(myCementType,2) * myfc28;
        double fcS = calcConcStrengthFactor(myCementType,ageS) * myfc28;

        double Ec0 = newDensityFactor * Math.sqrt(fc0); // square root of fc0
        double EcS = newDensityFactor * Math.sqrt(fcS); // square root of fcS

        double EeTS = EcS / (1 + myCreepTS) ;
        double EeAdjustTS = EcS / (1 + myageAdjustmentTS) ;
        double EeAdjustT0 = Ec0 / (1 + myageAdjustmentT0) ;

        double nS = myESteel / EcS;
        double neTS = myESteel / EeTS;
        double neAdjustTS = myESteel / EeAdjustTS;
        double neAdjustT0 = myESteel / EeAdjustT0;

        myMaterialsCalculated.addElement(new Double(EcS));
        myMaterialsCalculated.addElement(new Double(nS));
        myMaterialsCalculated.addElement(new Double(EeTS));
        myMaterialsCalculated.addElement(new Double(neTS));
        myMaterialsCalculated.addElement(new Double(EeAdjustTS));
        myMaterialsCalculated.addElement(new Double(neAdjustTS));
        myMaterialsCalculated.addElement(new Double(EeAdjustT0));
        myMaterialsCalculated.addElement(new Double(neAdjustT0));

        return myMaterialsCalculated;
    }



    /**
     * Selects the type of Concrete Strength Factor
     * @param c the type
```

```
     * @param i the day of the applied force
     * @return a double value for the Concrete Strength Factor
     */
    private double calcConcStrengthFactor(String c,int i){
      double value;
      String s = "A";
      String s2 = "a";
      // NEED TO FIX THIS
      if (c.equals(s) || c.equals(s2)){
        //System.out.println(" we hava a cement type of A");
        value = i / (4 +( .85 * i));}
      else
         value = i / (2.3 +( .92 * i));
      return value;
    }



    /**
     * Calculates the elastic delta for the particular column on the
     * particular day
     * @param column the base column in question
     * @param force the applied load causing the elastic delta
     * @param day the day for which we want to calculate delta.
     * @return a double value for the delta of the column due to
     * elastic response on the day required
     */
    private double calcElastic(Column column, Vector thisDayMaterial,
                           double force)
  {


    Material m = column.getMaterial();
    float mySteelArea = 0;

      if (m instanceof ACIConcrete){
        ACIConcrete thisMaterial= (ACIConcrete)m;
        mySteelArea = thisMaterial.getMetalReinforcingArea();

      }


      float myLength = column.getLength();
      Section thisSection = column.getSection();
      float myArea = thisSection.getArea();
      double myRatio = mySteelArea / 100;

      double myEcS =
Double.valueOf(thisDayMaterial.elementAt(0).toString()).doubleValue();
      double mynS =
Double.valueOf(thisDayMaterial.elementAt(1).toString()).doubleValue();
      // Float.valueOf(creepCoef.elementAt(0).toString()).floatValue();

      double elasticValue = force * myLength / (myArea * myEcS * (1 + (myRatio
* mynS)));
      //System.out.println(force + "force");
```

```
      //System.out.println(myLength + "myLength");
      //System.out.println(myArea + "myArea");
      //System.out.println( myEcS + " myEcS");
      //System.out.println(myRatio + "myRatio");
      //System.out.println( mynS + " mynS");
      return elasticValue;
    }



    /**
     * Calculates the creep delta for the particular column on the
     * particular day
     * @param column the base column in question
     * @param force the applied load causing the creep delta
     * @param day the day for which we want to calculate delta.
     * @return a double value for the delta of the column due to creep
     * response on the day required
     */
    private double calcCreep(Column column, Vector thisDayMaterial,
                      double force)
    {

      Material m = column.getMaterial();
      float mySteelArea = 0;

      if (m instanceof ACIConcrete){
        ACIConcrete thisMaterial= (ACIConcrete)m;
        mySteelArea =thisMaterial.getMetalReinforcingArea();

      }


      float myLength = column.getLength();
      Section thisSection = column.getSection();

      float myArea = thisSection.getArea();
      double myRatio = mySteelArea / 100;


      double mynS =
Double.valueOf(thisDayMaterial.elementAt(1).toString()).doubleValue();
      double myEeTS =
Double.valueOf(thisDayMaterial.elementAt(2).toString()).doubleValue();
      double myneTS =
Double.valueOf(thisDayMaterial.elementAt(3).toString()).doubleValue();
      double myEeAdjustTS =
Double.valueOf(thisDayMaterial.elementAt(4).toString()).doubleValue();
      double myneAdjustTS =
Double.valueOf(thisDayMaterial.elementAt(5).toString()).doubleValue();

      // Float.valueOf(creepCoef.elementAt(0).toString()).floatValue();

      double creepValue1 = force * myLength / (myArea * myEeTS * (1 + (myRatio
* mynS)));
```

```
      double creepValue2 = force * myLength * myRatio * (mynS - myneTS)   /
(myArea * myEeAdjustTS * (1 + (myRatio * mynS)) * (1 + (myRatio *
myneAdjustTS)));

      double creepValue = creepValue1 + creepValue2;
      //System.out.println(force + "force");
      //System.out.println(myLength + "myLength");
      //System.out.println(myArea + "myArea");
      //System.out.println(myEeTS  + "myEeTS ");
      //System.out.println(myRatio + "myRatio");
      //System.out.println( mynS + " mynS");
      //System.out.println(myneTS + "myneTS");
      //System.out.println(myEeAdjustTS + "myEeAdjustTS");

      //System.out.println("creep Value " + creepValue);
      return creepValue;
    }

    /**
     * Calculates the shrink delta for the particular column on the
     * particular day
     * @param column the base column in question
     * @param day the day for which we want to calculate delta.
     * @return a double value for the delta of the column due to
     * shrinkage response on the day required
     */
    private static double calcShrink(Column column, double shrinkageCoeff)
    {
      float myLength = column.getLength();
      double shrinkage = myLength * shrinkageCoeff * 1000;
      //System.out.println("shrinkage Value " + shrinkage);
      return shrinkage;
    }

    /**
     * Calculates the relaxation delta for the particular column on
     * the particular day
     * @param column the base column in question
     * @param day the day for which we want to calculate delta.
     * @return a double value for the delta of the column due to
     * relaxation response on the day required
     */
    private double calcRelax(Column column, double shrinkageCoeff,
                        Vector thisDayMaterial)
    {
      Material m = column.getMaterial();
      float mySteelArea = 0;

      if (m instanceof ACIConcrete){
        ACIConcrete thisMaterial= (ACIConcrete)m;
        mySteelArea = thisMaterial.getMetalReinforcingArea();

      }

      float myLength = column.getLength();
      Section thisSection = column.getSection();
```

```
        float myArea = thisSection.getArea();
        double myRatio = mySteelArea / 100;

        double myneAdjustT0 =
Double.valueOf(thisDayMaterial.elementAt(0).toString()).doubleValue();
        // Float.valueOf(creepCoef.elementAt(0).toString()).floatValue();

        double relax = 0 - (1000 * shrinkageCoeff * myLength * myRatio *
myneAdjustT0 /  (1 + (myRatio * myneAdjustT0)));
        //System.out.println("relax Value " + relax);
        //System.out.println(shrinkageCoeff + "shrinkageCoeff");
        //System.out.println(myLength + "myLength");
        //System.out.println(myRatio + "myRatio");
        //System.out.println(myneAdjustT0 + "myneAdjustT0");


        return relax;
    }
}
```

```
/**
 * @(#)Rounding.java
 * Class to round double precision numbers and return a string
 *
 * History: Mar 2002 Copied from the net by G.Lewis
 *
 */
package structuralAnalyser;
import java.lang.Math;
import java.io.*;

public class Rounding
{
  public static String toString (double d, int place)
  {
    if (place <= 0)
      return ""+(int)(d+((d > 0)? 0.5 : -0.5));
    String s = "";
    if (d < 0)
      {
      s += "-";
      d = -d;
      }
    d += 0.5*Math.pow(10,-place);
    if (d > 1)
      {
      int i = (int)d;
      s += i;
      d -= i;
      }
    else
      s += "0";
    if (d > 0)
      {
      d += 1.0;
      String f = ""+(int)(d*Math.pow(10,place));
      s += "."+f.substring(1);
      }
    return s;
  }

}
```

```
/*
 * @(#)ShrinkageAlgorithm.java
 *
 * History:   Mar 2001: Written by M Gardner.
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

import structuralAnalyser.structuralElement.*;
import structuralAnalyser.material.*;

import java.util.*;
import java.io.*;
import java.lang.Math.*;
import Jama.*; //matrix package, from http://math.nist.gov/javanumerics/jama/

/**
 * This is an abstract class contains for calculating the shrinkage of
 * columns
 */
public abstract class ShrinkageAlgorithm{

    /**
     * Calculate delta (shrinkage) for a set of columns
     * @param columns the columns to calculate the shrinkage for
     * @param day the day to calculate the shrinkage on.
     *
     */
    public abstract  Vector calcColumnDelta(Vector columns, int day)
      throws InvalidParameterException;

}
```

```
/*
 * @(#)StructuralAnalyser.java
 *
 * History:  1999-2000: Written by M Gardner and C. Dalton
 *           Oct 2001:  G.Lewis added comments, deleted obsolete code,
 *                and changed variable names.
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import java.io.*;
import structuralAnalyser.*;
import structuralAnalyser.structuralElement.*;
import structuralAnalyser.material.*;

/**
 * This class serves contains the main method for the structural analyser
 */
public class StructuralAnalyser
{
  /** This class should never be constructed */
  private StructuralAnalyser(){}

  /**The main method */
  public static void main(String[] args) {
    AnalyserFrame frame = new AnalyserFrame();
    frame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {System.exit(0);}
    });
    //    frame.setSize(500, 250);
    frame.setVisible(true);

    /*
    try{
      Column c2 = new Column(concrete,50,section);
      Column c3 = new Column(concrete,50,section);

      //Add the columns to the structural environment
      Structure structure = new Structure();

      structure.addElement(c1,1,new Coordinate(0,0,0));
      structure.addElement(c2,2,new Coordinate(0,50,0));
      structure.addElement(c3,5,new Coordinate(0,100,0));

      //add some loads to the columns
      Load l1 = new Load(c1);
      Load l2 = new Load(c2);
      Load l3 = new Load(c3);
      l1.setWeight(1000);
      l2.setWeight(2000);
```

```
        l3.setWeight(3000);

        structure.addLoad(l1,5,c1);
        structure.addLoad(l2,7,c2);
        structure.addLoad(l3,9,c2);

        //calculate the shrinkage on column 1 at day 150
        double shrinkage = calcColumnDelta(c1,structure,150);
        System.out.println("The shrinkage on column 1 at day 150 is " +
                    shrinkage + ".");
      }
      catch (InvalidPositionException e){
        System.err.println(e.getMessage());
        System.exit(-1);
      }
      catch (InvalidLoadException e){
        System.err.println(e.getMessage());
        System.exit(-1);
      }
      catch (InvalidParameterException e){
        System.err.println(e.getMessage());
        System.exit(-1);
        }*/
    }
}
```

```
/*
 * @(#)StructuralElement.java
 *
 * History:  1999-2000: Written by M Gardner and C. Dalton
 *           Oct 2001:  Re-written by G. Lewis
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser.structuralElement;

import java.io.*;
import structuralAnalyser.*;
import structuralAnalyser.material.*;

/**
 * Abstract base class for all structures. Subclasses include Columns,
 * Floors, Beams, etc.
 */
public abstract class StructuralElement implements Serializable
{
  /** The id of the element */
  protected int myId;
  /**The material this structure is made of */
  protected Material myMaterial;
  /**The weight of the structure */
  protected float myWeight;
  /**The starting coordinate of the structure */
  protected Coordinate myStartCoordinate;
  /**The day the element was added to the structure*/
  protected int myDayConstructed;

  /**Construct the structure*/
  public StructuralElement(int id, Material material)
    throws InvalidParameterException
  {
    setId(id);
    setMaterial(material);
  }

  /** Get the id of this element */
  public int getId()
  {
    return myId;
  }

  /** Set the id of this element
   * @param id the id to set it to
   */
  public void setId(int id) throws InvalidParameterException
  {
    if (id < 0)
      throw new InvalidParameterException("Attempting to set theid to less
than 0");
    myId = id;
  }
```

```
  /** Get the material of this structure */
  public Material getMaterial()
  {
    return myMaterial;
  }

  /** Set the material of this element
   * @param m the material to set it to
   */
  public void setMaterial(Material material) throws InvalidParameterException
  {
    if (material == null)
      throw new InvalidParameterException("Attempting to set the material to
null");
    myMaterial = material;
  }

  /** Get the weight of this structural element */
  public float getWeight()
  {
    return myWeight;
  }

  /** Set the material of this element
   * @param m the material to set it to
   */
  public void setWeight(float weight) throws InvalidParameterException
  {
    if (weight <= 0)
      throw new InvalidParameterException("Attempting to set the weight of an
element to less than or equal to 0");
           myWeight = weight;
  }

  /** Get the starting coordinate of the element */
  public Coordinate getStartCoordinate ()
  {
    return myStartCoordinate;
  }

  /**Set the starting coordinate of the element
   * @param coord the position to be set
   */
  public void setStartCoordinate (Coordinate startCoordinate)
  throws InvalidParameterException
  {
    if (startCoordinate == null)
      throw new InvalidParameterException("Attempting to set the start
coordinate to  null");
    myStartCoordinate = startCoordinate;
  }


  /** Get the day this element was added to the structure */
  public int getDayConstructed()
  {
```

```
    return myDayConstructed;
  }

  /**Set the day this element was added to the structure
   * @param dayConstructed the day this element was added to the structure
   */
  public void setDayConstructed(int dayConstructed) throws
InvalidParameterException
  {
    if (dayConstructed < 0)
      throw new InvalidParameterException("Attempting to set the day a element
was constructued to less than 0");
    myDayConstructed = dayConstructed;
  }

  public abstract String getToolTipText();

}
```

```
/*
 * @(#)Structure.java
 *
 * History:  1999-2000: Written by M Gardner and C. Dalton
 *           Oct 2001:  Rewritten from scratch by G.Lewis.
 *         Mar 2002: Modified for framing input by G.Lewis.
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */

package structuralAnalyser;

import structuralAnalyser.structuralElement.*;
import structuralAnalyser.material.*;

import java.util.*;
import java.io.*;
import java.lang.Math.*;
import Jama.*; //matrix package, from http://math.nist.gov/javanumerics/jama/

/** Structural elements (e.g. Columns) are added to the structure. The
 structure is a singleton.
*/
public class Structure implements Serializable {
  /**Serial version ID for backwards compatability when reading objects */
  static final long serialVersionUID = -2833354243440995289L;

  /**Unique id for each new element of the structure */
  private int myNextId = 0;

  /** The singleton instance of this class */
  static private Structure myInstance  = null;

  /**Have any changes been made since the last save? */
  transient private boolean myStructureChanged = false;

  /**A vector of materials that can be used in this structure */
  private Vector myMaterials;

  /**A vector of sections that can be used in this structure */
  private Vector mySections;

  /**A sorted set of Integers indicating which days construction
     has been done on */
  private SortedSet myConstructionDays;

  /**The structual elements of this structure*/
  private Hashtable myElements;

  /**A sorted set of Integers indicating which days loads have been added.*/
  private SortedSet myLoadDays;

  /**The loads of this structure*/
  private Hashtable myLoads;

  /**A sorted set of Integers indicating which days shrinkages have been
```

```
   calculated (when framing is not included) */
private SortedSet myNonFramingShrinkageDays;

/**A sorted set of Integers indicating which days shrinkages have been
   calculated (when framing is included) */
private SortedSet myFramingShrinkageDays;


/**The shrinkages of this structure when framing has not been
 * included in the calculation*/
private Hashtable myNonFramingShrinkages;


/**The shrinkages of this structure when framing has been
 * included in the calculation*/
private Hashtable myFramingShrinkages;

  /** The framing added to the structure, stored as a hashtable.
   * The index of the hashtable is the floor number for the framing
   * and the framing is stored as a matrix.
   */
  private Hashtable myFramings;


/**Constructor */
public Structure()
{
  myMaterials = new Vector();
  mySections = new Vector();
  myConstructionDays = new TreeSet(new intComparator());
  myElements = new Hashtable();
  myLoadDays = new TreeSet(new intComparator());
  myLoads = new Hashtable();
  myNonFramingShrinkageDays = new TreeSet(new intComparator());
  myNonFramingShrinkages = new Hashtable();
  myFramingShrinkageDays = new TreeSet(new intComparator());
  myFramingShrinkages = new Hashtable();
  myFramings = new Hashtable();
}

/** Get the instance of this class
 */
public static Structure getInstance(){
  if (myInstance ==  null)
    myInstance = new Structure();
  return myInstance;
}

/** Set the instance of this class */
public static void setInstance(Structure instance){
  if (instance != null)
    myInstance = instance;
}

public int getNextUniqueId(){
  return myNextId++;
}
```

```
    /**Get the number of floors in this structure */
    public SortedSet getFloorHeights(){
      Vector allElements = getAllElements();
      SortedSet floors = new TreeSet(new dblComparator());
      for (int i = 0; i< allElements.size(); i++){
          if (allElements.elementAt(i) instanceof Column){
            Column col = (Column) allElements.elementAt(i);
            Coordinate startCoord = col.getStartCoordinate();
            floors.add(new Double(startCoord.getYPosition()));
            floors.add(new Double(startCoord.getYPosition() +
col.getLength()));
          }
      }
      return floors;
    }


    /**Get the construct stages. That is, the unique days when the
     * structure changed.*/
    public SortedSet getStages(){
      Vector allElements = getAllElements();
      SortedSet stages = new TreeSet(new intComparator());
      stages.addAll(myConstructionDays);
      stages.addAll(myLoadDays);

      int numFloors = getNumFloors();
      for (int i =1; i < numFloors; i++){
          Framing framing = getFraming(new Integer(i));
          Integer day = new Integer(framing.getDayApplied());
          stages.add(day);
      }

      stages.addAll(myLoadDays);
      return stages;
    }


    /** Get the height of a given floor */
    private double getFloorHeight(Integer floorNumber){
      SortedSet floorHeights = getFloorHeights();
      return ((Double)
floorHeights.toArray()[floorNumber.intValue()]).doubleValue();
    }

    /** Get the number of columns on a given floor*/
    public int getNumColumns(Integer floorNumber){
      return getColumns(floorNumber).size();
    }

    /**Get the number of floors of the structure. The roof is included
     * in this.*/
    public int getNumFloors(){
      SortedSet floorHeights = getFloorHeights();
      return floorHeights.size();
    }
```

```
    /** Get the columns on a given floor
     * @param floorNumber the floor to get the columns for
     */
    public Vector getColumns(Integer floorNumber){
      Vector allElements = getAllElements();
      Vector columns = new Vector();
      for (int i = 0; i< allElements.size(); i++){
          if (allElements.elementAt(i) instanceof Column){
            Column col = (Column) allElements.elementAt(i);
            Coordinate startCoord = col.getStartCoordinate();
            float startHeight = startCoord.getYPosition();
            double floorHeight = getFloorHeight(floorNumber);
            if ((startHeight - floorHeight > - 0.01) && (floorHeight -
startHeight < 0.01) && (startHeight - floorHeight < 0.01))
                columns.add(col);
          }
      }
      return columns;
    }


  /**Add a material to the materials that can be used */
  public void addMaterial(Material material){
    myMaterials.add(material);
  }

  /**Get the materials that can be used in this structure */
  public Vector getMaterials(){
    return myMaterials;
  }

  /**Add a section to the sections that can be used */
  public void addSection(Section section){
    mySections.add(section);
  }

  /**Get the sections that can be used in this structure */
  public Vector getSections(){
    return mySections;
  }

  /**Get whether any changes have been made since the last save?*/
  public boolean getStructureChanged(){
    return myStructureChanged;
  }

  /**Set whether any changed have been made since the last save*/
  public void setStructureChanged(boolean changed){
    myStructureChanged = changed;
  }

  /**Add an element to the structure on a given day
   * @param element the  structural element to be added.
   * @param day the day the structure was added to this structure
   * @param startCoordinate the starting coordinate of the element in this
   * structure
   */
```

```
  public void addElement(StructuralElement element, int day,
                Coordinate startCoordinate)
                throws InvalidPositionException,
                     InvalidParameterException{
    if (!canAddElement(element,day,startCoordinate))
      throw new InvalidPositionException("The element cannot be added at that
position in the structure.");

    try{
    String weight = Float.toString(element.getWeight());
    double thisWeight = Double.valueOf(weight).doubleValue();
    Load thisLoad = new Load(element,day,thisWeight,true);

    addLoad(thisLoad,day);
    }
    catch(InvalidLoadException e){
      System.err.println(e.getMessage());
      // JOptionPane.showMessageDialog(LoadsDialog.this,
      //"Cannot add load",
      //"Error",
      //JOptionPane.ERROR_MESSAGE);
      }
      catch(InvalidParameterException e){
        System.err.println(e.getMessage());
        System.exit(-1);
      }

    element.setStartCoordinate(startCoordinate);
    element.setDayConstructed(day);
    Integer theDay = new Integer(day);
    myConstructionDays.add(theDay);
    Vector theDayElements = (Vector) myElements.get(theDay);
    if (theDayElements == null){
      theDayElements = new Vector();
      myElements.put(theDay,theDayElements);
    }
    theDayElements.add(element);
    setStructureChanged(true);
  }


  /**Remove an element from the structure
   * @param element the  structural element to be removed.
   */
  public void removeElement(StructuralElement element) {
    Integer theDay = new Integer(element.getDayConstructed());
    Vector theDayElements = (Vector) myElements.get(theDay);
    boolean removed = theDayElements.removeElement(element);
    Assert.assert(removed);
    //remove the construction day if necessary
    if (theDayElements.size() == 0){
      myConstructionDays.remove(new Integer(element.getDayConstructed()));
      setStructureChanged(true);
    }
  }

  /**Get the elements added to this structure on a given day
```

```
  * @param day the day to get the elements on
  * @return a vector of the element added to this structure on
  * the given day, or null if no elements have been added on thay day.
  */
private Vector getElements(int day){
  Integer theDay = new Integer(day);
  return (Vector) myElements.get(theDay);
}

/**Get all the elements added to this structure
 * @return a vector of the elements added to this structure. The
 * vector is empty if no elements are in the structure.
 */
public Vector getAllElements(){
  if (myConstructionDays.size() == 0)
    return new Vector();
  int day = ((Integer) myConstructionDays.last()).intValue();
  return getAllElements(day);
}

/**Get all the elements added to this structure up to and
 * including a given day
 * @param day the day to get the elements added up to
 * @return a vector of the elements added to this structure up to
 * the given day, or null if no elements have been added on thay day.
 */
private Vector getAllElements(int day){
  Vector allElements = new Vector();
  Iterator iter = myConstructionDays.iterator();

  while (iter.hasNext()){
    Integer constructionDay = (Integer) iter.next();
    if (constructionDay.intValue() <= day)
    allElements.addAll(getElements(constructionDay.intValue()));
    else
    break;
  }
  return allElements;
}

/**Get all the elementss added to this structure from a given day
 * up to and including a given day
 * @param day1 the day to get the elements added from
 * @param day2 the day to get the elements added up to
 * @return a vector of the elements added to this structure up to
 * the given day, or null if no elements have been added on thay day.  */
private Vector getAllElements(int day1, int day2){
  Vector allElements = new Vector();
  Iterator iter = myConstructionDays.iterator();

  while (iter.hasNext()){
    Integer constructionDay = (Integer) iter.next();
    if ((constructionDay.intValue() >= day1) &&
      (constructionDay.intValue() <= day2))
    allElements.addAll(getElements(constructionDay.intValue()));
  }
  return allElements;
```

```
  }


  /**Check whether an element can be added at a given that a given
   * element can be added at a given position
   * @param element the element to be added
   * @param pos the coordinate the element is to be added at
   */
  private boolean canAddElement(StructuralElement element, int day,
                          Coordinate pos){
    if (element instanceof Column){
      Column column = (Column) element;
      return canAddColumn(column,day,pos);
    }
    //need to check adding of other elements
    System.out.println("Structure.canAddElement is not implemented for Beams,
Floors and Walls.");
    return true;
  }

  /**Check whether an element can be added at a given position on a given day
   * @param column the column to be added
   * @param day the day at which the column is to be added
   * @param pos the coordinate the element is to be added at
   */
  private boolean canAddColumn(Column column, int day, Coordinate pos){
    //first check all elements added to see if there is a column in
    //this position
    // System.out.println(" Here in testin add column ");
    Vector allElements = getAllElements();
    if (allElements == null)
      return (pos.getYPosition() == 0);
    //check if there is a column already in that position
    for (int i = 0; i < allElements.size(); i++){
      StructuralElement existingElt =
      (StructuralElement) allElements.elementAt(i);
      if (existingElt instanceof Column){
      Coordinate startCoord = existingElt.getStartCoordinate();
      if (startCoord.isEqual(pos)){
        //System.out.println(" Here in test height part 1");
          return false;
      }
      }
    }

    //check if there are columns to support the column being added
    Vector theElements = getAllElements(day);
    Column highestColumn = null;
    for (int i = 0; i < theElements.size(); i++){
      StructuralElement existingElt =
      (StructuralElement) theElements.elementAt(i);
      if (existingElt instanceof Column){
      Coordinate startCoord = existingElt.getStartCoordinate();
      //check if there is a column already in that position
      if (startCoord.isEqual(pos)){
        //    System.out.println(" Here in test height part 2");
        return false;
```

```
      }
      if((startCoord.getXPosition() == pos.getXPosition()) &&
         (startCoord.getZPosition() == pos.getZPosition())){
        //System.out.println(" We have equal x and z coords");
        if ((highestColumn == null) ||
            (startCoord.getYPosition() >
             highestColumn.getStartCoordinate().getYPosition()))
          highestColumn = (Column) existingElt;
      }
      }
    }
    if (highestColumn == null)
      return (pos.getYPosition() == 0);
    //System.out.println( highestColumn.getStartCoordinate().getYPosition() +
highestColumn.getLength() + " col + Length");
    // System.out.println(pos.getYPosition() + " position");
    if((highestColumn.getStartCoordinate().getYPosition() +
      highestColumn.getLength() -  pos.getYPosition() < 0.001)
&&(highestColumn.getStartCoordinate().getYPosition() +
highestColumn.getLength() -  pos.getYPosition() > -0.001)){
                        //System.out.println(" Here in test height part 3");
      return true;
    }
    //System.out.println(" Here in test height part 4");

    return false;
  }


  /**Check whether an element can be removed from the structure
   * @param element the element to be removed
   */
  public boolean canRemoveElement(StructuralElement element){
    if (element instanceof Column){
      Column column = (Column) element;
      return canRemoveColumn(column);
    }
       System.out.println("Structure.canRemoveElement is not implemented for
Beams, Floors and Walls.");
    return true;
  }


  /**Check whether a column can be removed from the structure
   * @param column the column to be removed
   */
  private boolean canRemoveColumn(Column column){
    //check all elements to see if this column supports any others
    Vector allElements = getAllElements();
    Assert.assert(allElements != null);
    Coordinate columnCoord = column.getStartCoordinate();
    for (int i = 0; i < allElements.size(); i++){
      StructuralElement existingElt =
      (StructuralElement) allElements.elementAt(i);
      if (existingElt instanceof Column){
      Coordinate existingCoord = existingElt.getStartCoordinate();
      if ((existingCoord.getXPosition() == columnCoord.getXPosition()) &&
```

```
        (existingCoord.getYPosition() > columnCoord.getYPosition()) &&
        (existingCoord.getZPosition() == columnCoord.getZPosition()))
      return false;
    }
  }
  return true;
}


/**Add a load to the structure on a given day
 * @param load the load to add to the structure.
 * @param day the day the load was added to this structure
 */
public void addLoad(Load load, int day)
                throws InvalidLoadException{
  if (!canAddLoad(load,day))
    throw new InvalidLoadException("The load cannot be added");
  Integer theDay = new Integer(day);
  myLoadDays.add(theDay);
  Vector theDayLoads = (Vector) myLoads.get(theDay);
  if (theDayLoads == null){
    theDayLoads = new Vector();
    myLoads.put(theDay,theDayLoads);
  }
  theDayLoads.add(load);
  setStructureChanged(true);
}

/**Remove a load from the structure
 * @param load the load to remove.
 */
public void removeLoad(Load load){
  Vector loads = getLoads(load.getDayConstructed());
  boolean removed = loads.remove(load);
  Assert.assert(removed);
  setStructureChanged(true);
}

/**Get the loads added to this structure on a given day
 * @param day the day to get the elements on
 * @return a vector of the load added to this structure on
 * the given day, or null if no elements have been added on thay day.
 */
private Vector getLoads(int day){
  Integer theDay = new Integer(day);
  return (Vector) myLoads.get(theDay);
}

/**Get all the loads added to this structure up to and
 * including a given day
 * @param day the day to get the loads added up to
 * @return a vector of the loads added to this structure up to
 * the given day, or null if no loads have been added on thay day.
 */
private Vector getAllLoads(int day){
  Vector allLoads = new Vector();
  Iterator iter = myLoadDays.iterator();
```

```
    while (iter.hasNext()){
      Integer loadDay = (Integer) iter.next();
      if (loadDay.intValue() <= day)
      allLoads.addAll(getLoads(loadDay.intValue()));
      else
      break;
    }
    return allLoads;
  }

  /**Get all the loads added to this structure from a given day
   * up to and including a given day for a given column
   * @param day1 the day to get the loads added from
   * @param day2 the day to get the loads added up to
   * @param column the column we are getting loads for
   * @return a vector of the loads added to this structure up to
   * the given day, or null if no loads have been added on thay day.  */
  public Vector getAllLoads(int day1, int day2, Column column){
    //System.out.println("day1" +day1);
    //System.out.println("day2" +day2);
    Vector allLoads = new Vector();
    Iterator iter = myLoadDays.iterator();
    //System.out.println("myLoadDays" +myLoadDays);
    while (iter.hasNext()){
      Integer loadDay = (Integer) iter.next();
      //System.out.println("loadDay "+loadDay);
      if ((loadDay.intValue() >= day1) &&
        (loadDay.intValue() <= day2)){
      //System.out.println("this is day1 "+day1);
      Vector loads = getLoads(loadDay.intValue());
      //System.out.println("loads "+loads);
      for (int i =0; i < loads.size(); i++){
        Load load = (Load) loads.elementAt(i);
        if (load.getAppliedElement() == column)
          allLoads.add(load);
      }
      }
       else if((loadDay.intValue() >= day2))
      break;
    }
    return allLoads;
  }


/**Get all the loads added to this structure on a given day for a given column
   * @param day1 the day to get the loads added from
   * @param column the column we are getting loads for
   * @return a vector of the loads added to this structure up to
   * the given day, or null if no loads have been added on thay day.  */
  private Vector getAllLoads(int day1, Column column){
    Vector allLoads = new Vector();
    Vector loads = getLoads(day1);
    //System.out.println(loads);
    if (loads != null)
      {
      for (int i =0; i < loads.size(); i++)
```

```
      {
      Load load = (Load) loads.elementAt(i);
      if (load.getAppliedElement() == column)
        allLoads.add(load);
      }
    }
  //System.out.println(allLoads);
  return allLoads;
}


/**Get all the loads added to this structure that apply to a given element
 * @param element the element the loads apply to
 * @return a vector of the loads added to this structure that apply
 * to the given element
 */
public Vector getAllLoads(StructuralElement element){
  Vector allLoads = new Vector();
  Iterator iter = myLoadDays.iterator();

  while (iter.hasNext()){
    Integer loadDay = (Integer) iter.next();
    Vector loads = getLoads(loadDay.intValue());
    for (int i =0; i < loads.size(); i++){
    Load load = (Load) loads.elementAt(i);
    if (load.getAppliedElement() == element)
      allLoads.add(load);
    }
  }
  return allLoads;
}


/**Check whether a load can be added to a given element on a given day
 * @param element the element to be added
 * @param pos the coordinate the element is to be added at
 */
private boolean canAddLoad(Load load, int day){
  if (day < load.getAppliedElement().getDayConstructed())
    return false;
  return true;
}

  /** A inner-wrapper-class to combine the column and its shrinkage */
  public class ColumnAndShrinkage implements Serializable {
    /** the column */
    private Column myColumn;
    /** the shrinkage of the column (-1 if not set). */
    private double myShrinkage = -1;


    /**Constructor
     *@param column the column
     */
    public ColumnAndShrinkage(Column column){
```

```
        myColumn = column;
    }

    /**Set the shrinkage */
    public void setShrinkage(double shrinkage){
        myShrinkage = shrinkage;
    }

    /** Get the shrinkage */
    public double getShrinkage(){
        return myShrinkage;
    }

    /** Get the column */
    public Column getColumn(){
        return myColumn;
    }
  }

/**Add a shrinkage to the structure on a given day
 * @param day the day the shrinkage was calcuated for
 * @param column the column the shrinkage is for
 * @param shrinkage the shrinkage of the column
 * @param framing if true then the shrinkage was calculated with framing
 */
public void addShrinkage(int day, Column column, double shrinkage,
                  boolean framing){
    Integer theDay = new Integer(day);
    Vector theDayShrinkages = null;
    if (framing){
      myFramingShrinkageDays.add(theDay);
      theDayShrinkages =  (Vector) myFramingShrinkages.get(theDay);
    }
    else{
      myNonFramingShrinkageDays.add(theDay);
      theDayShrinkages =  (Vector) myNonFramingShrinkages.get(theDay);
    }

  if (theDayShrinkages == null){
    theDayShrinkages = new Vector();
    //add each column to the shrinkages
    Vector allElements = getAllElements();
    for (int i = 0; i< allElements.size(); i++){
      if (allElements.elementAt(i) instanceof Column){
          Column col = (Column) allElements.elementAt(i);
          ColumnAndShrinkage cas = new ColumnAndShrinkage(col);
          theDayShrinkages.add(cas);
      }
    }
    if (framing)
      myFramingShrinkages.put(theDay,theDayShrinkages);
    else
      myNonFramingShrinkages.put(theDay,theDayShrinkages);
  }

  for (int i = 0; i < theDayShrinkages.size(); i++){
```

```
      ColumnAndShrinkage cas = (ColumnAndShrinkage)
theDayShrinkages.elementAt(i);
      if (cas.getColumn() == column){
          cas.setShrinkage(shrinkage);
      }
    }
    setStructureChanged(true);
  }

    /**Get the shrinkages for a given day
     * @param day the day to get the shrinkages on
     * @param framing if true then get the shrinkages calculated with framing
     */
    public Vector getShrinkages(int day, boolean framing){
      Integer theDay = new Integer(day);
      Vector casVector = null;
      if (framing)
          casVector = (Vector) myFramingShrinkages.get(theDay);
      else
          casVector = (Vector) myNonFramingShrinkages.get(theDay);

      Vector shrinkages = new Vector();
      for (int i = 0; i < casVector.size(); i++){
          ColumnAndShrinkage cas = (ColumnAndShrinkage)
casVector.elementAt(i);
          if (cas.getShrinkage() >= 0)
            shrinkages.addElement(new Double(cas.getShrinkage()));
          else
            shrinkages.addElement(null);
      }
      return shrinkages;
    }

    /** Get all the days that shrinkages have been calculated for
     * @param framing if true then get the shrinkages days when
     * framing been tajen into account
     */
    public SortedSet getShrinkageDays(boolean framing){
      if (framing)
          return myFramingShrinkageDays;
      else
          return myNonFramingShrinkageDays;
    }

    /** Get the framings for a given floor
     * @param floorNumber the floor to get the framings for
     */
    public Framing getFraming(Integer floorNumber){
      //System.out.println(" here in getFraming");
      Framing framing = (Framing) myFramings.get(floorNumber);
      if (framing == null){
          framing = new Framing(floorNumber.intValue(),
                          getNumColumns(floorNumber));
          myFramings.put(floorNumber,framing);
      }
      else{
          //check whether the structure has changed since the framing
```

```
            //was entered
            if (!framing.isDataValid()){
              myFramings.remove(floorNumber);
              framing = new Framing(floorNumber.intValue(),
                                    getNumColumns(floorNumber));
              myFramings.put(floorNumber,framing);
            }
        }
        return framing;
    }

    /**Set the framing for a given floor
     * @param floorNumber the floor number
     * @param framing the framing
     */
    public void setFraming(Integer floorNumber, Framing framing){
      myFramings.remove(floorNumber);
      myFramings.put(floorNumber,framing);
      setStructureChanged(true);
    }

  /**Get the force on a column for a given day
   *@param day the day to get the force on.
   */
  public float getForce(Column column, int day)
  {
    //start with the weight of the column
    //System.out.println("here on day " + day + " and column " + column);
    int dayConstructed = column.getDayConstructed();
    float force = 0;
    if (dayConstructed <= day)
      {
      if (dayConstructed == day)
          force = column.getWeight();

          //add the loads on the column from the day the column was
          //constructed up until the given day
       Vector loads = getAllLoads(day, column);
       for(int i = 0; i < loads.size(); i++)
         force += ((Load) loads.elementAt(i)).getWeight();
      }

    return force;
  }



/**
   * Gets the column under a particular column. returns null if the bottom
column.
   * @param column the column in question
   * @return the column under the column in question
   */

  public Column getColumnUnder(Column column)
  {
```

```
//The following didn't make sense to me. GAL 10/4/2003.

//     Coordinate columnCoord = column.getStartCoordinate();
//     if(columnCoord.getYPosition() == 0)
//       return null;
//
//     Vector possibleColumns = new Vector();
//
//     Vector allElements = getAllElements();
//     Assert.assert(allElements != null);
//
//     for (int i = 0; i < allElements.size(); i++){
//       StructuralElement existingElt =
//    (StructuralElement) allElements.elementAt(i);
//       if (existingElt instanceof Column){
//    Coordinate existingCoord = existingElt.getStartCoordinate();
//    if ((existingCoord.getXPosition() == columnCoord.getXPosition()) &&
//        (existingCoord.getYPosition() <  columnCoord.getYPosition()) &&
//        (existingCoord.getZPosition() == columnCoord.getZPosition()))
//      possibleColumns.addElement(existingElt);
//       }
//     }
//
//     Column columnToReturn = (Column)possibleColumns.elementAt(0);
//     for (int i=0; i < possibleColumns.size(); i++){
//       Column aColumn = (Column)possibleColumns.elementAt(i);
//       Coordinate coordToReturn = columnToReturn.getStartCoordinate();
//       Coordinate otherCoord = aColumn.getStartCoordinate();
//       if(otherCoord.getYPosition() >  coordToReturn.getYPosition())
//    columnToReturn = aColumn;
//     }
//
//     return columnToReturn;

//Try this instead.

      Coordinate columnCoord = column.getStartCoordinate();
      if(columnCoord.getYPosition() == 0)
        return null;

      Vector allElements = getAllElements();
      Assert.assert(allElements != null);

      Column columnToReturn = null;
      for (int i = 0; i < allElements.size(); i++){
        StructuralElement existingElt =
            (StructuralElement) allElements.elementAt(i);
        if (existingElt instanceof Column){
            Coordinate existingCoord = existingElt.getStartCoordinate();
            if (existingCoord.getYPosition() > columnCoord.getYPosition())
              continue; //column not underneath given column

            if ((existingCoord.getXPosition() == columnCoord.getXPosition())
&&
              (existingCoord.getZPosition() == columnCoord.getZPosition()) &&
              (existingCoord.getYPosition() < columnCoord.getYPosition())){
              if ((columnToReturn == null) ||
```

```
                    (columnToReturn.getStartCoordinate().getYPosition() <
                     existingCoord.getYPosition()))
                    columnToReturn = (Column) existingElt;
                }
            }
        }
        return columnToReturn;
    }


    /**
     * Get all the column elements that are constructed on top of the
     * base column up until a given day.
     * @param column the base column
     * @param day columns constructed after this day will not appear in the
stack
     * @return a vector containing each column on top of the base column
     * including the base column
     */
    public Vector getColumnStack(Column column, int day)
    {
      Vector elements = getAllElements(column.getDayConstructed(),day);
      Vector columnStack = new Vector();
      // columnStack.addElement(column);

      Coordinate baseCoord = column.getStartCoordinate();

      for (int i = 0; i < elements.size(); i++){
        StructuralElement existingElt =
        (StructuralElement) elements.elementAt(i);
        if (existingElt instanceof Column){
        Coordinate startCoord = existingElt.getStartCoordinate();

        if((startCoord.getXPosition() == baseCoord.getXPosition()) &&
           (startCoord.getZPosition() == baseCoord.getZPosition())){
          columnStack.addElement(existingElt);
        }
        }
      }
      return columnStack;
    }



    /**
     * Sort a list of column in order as a list of columns their base columns

     * @param columns the list of columns to sort

     * @return a vector containing a vector with each column on top of each of
the base columns
     * including the base columns
     */
    public Vector getSortedStack(Vector columns)
    {

      Vector columnStacks  = new Vector();
```

```
    //FIND ALL THE COLUMNS WITH Y COOD = 0;
    // FIND THE SORTED STACK FOR EACH COLUMN AND RETURN A VECTOR OF VECTORS
    Vector baseColumns  = new Vector();
    Column firstColumn = (Column)columns.elementAt(0);
    Coordinate baseCoord =firstColumn.getStartCoordinate();
    //System.out.println("baseCoord " + baseCoord);
//baseColumns.addElement((Column)columns.elementAt(0));
 baseColumns.addElement(firstColumn);
    for (int i = 1; i < columns.size(); i++){
     Column  anElt = (Column) columns.elementAt(i);
     //System.out.println("anElt " + anElt);
      Coordinate startCoord = anElt.getStartCoordinate();
      //System.out.println("startCoord " + startCoord);
      // if((startCoord.getXPosition() != baseCoord.getXPosition()) ||
(startCoord.getZPosition() != baseCoord.getZPosition()) &&
(startCoord.getYPosition() == baseCoord.getYPosition())){
 if(startCoord.getYPosition() == baseCoord.getYPosition()){
      baseColumns.addElement((Column)columns.elementAt(i));
      }
    }
    for (int j = 0; j < baseColumns.size(); j++){
      Vector aStack = new Vector();
      Column aBaseColumn = (Column)baseColumns.elementAt(j);
      aStack.addElement(aBaseColumn);
      Coordinate aBaseCoord =aBaseColumn.getStartCoordinate();
      for (int k = 0; k < columns.size(); k++){
       Column anElement = (Column)columns.elementAt(k);
       Coordinate anElementStartCoord = anElement.getStartCoordinate();
        if((anElementStartCoord.getXPosition() == aBaseCoord.getXPosition())
&& (anElementStartCoord.getZPosition() == aBaseCoord.getZPosition()) &&
(anElementStartCoord.getYPosition() != aBaseCoord.getYPosition())){
          aStack.addElement((Column)anElement);
        }
      }
      columnStacks.addElement(aStack);
      }


    return columnStacks;
  }



}
```

```
/*
 * @(#) TableDialog.java
 *
 * History:   Oct 2000: Written by M.Gardner.
 *
 * Copyright (C) 2001 Mark Gardner
 *
 */
package structuralAnalyser;

import javax.swing.*;
import javax.swing.table.*;
import javax.swing.border.*;
import javax.accessibility.*;

import java.io.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;

import structuralAnalyser.structuralElement.*;
import structuralAnalyser.material.*;

/** Class to display the shrinkages
   */
public class TableDialog extends JDialog {
    /** the table to be displayed */
    JTable myTable = null;
    /** the data model for the table */
    DefaultTableModel tableModel = null;
    /**Save as button */
    JButton mySaveAsBtn = null;
    /** Save As action */
    Action mySaveAsAction = new SaveAsAction();
    /**The path to the file for saving*/
    String myFilePath;

    /** constructor
     * @param the parent of this dialog */
  public TableDialog(JDialog parent) {
    super(parent,"Results Table",true);
    setup();
  }

    /** constructor
     * @param the parent of this dialog */
  public TableDialog(JFrame parent) {
    super(parent,"Results Table",true);
    setup();
  }

    /**set up table and display the table*/
    private void setup(){
      //add the save as button
      mySaveAsBtn = new JButton("<html>Save As Text ...</html>");
      mySaveAsBtn.addActionListener(mySaveAsAction);
      mySaveAsBtn.setToolTipText("Click to save the results as a text file.");
```

```
    getContentPane().add(mySaveAsBtn,BorderLayout.NORTH);

    tableModel = new DefaultTableModel();
    myTable = new JTable(tableModel);
    myTable.setPreferredScrollableViewportSize(new Dimension(300, 70));
    myTable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);


    //Create the scroll pane and add the table to it.
    JScrollPane scrollPane = new JScrollPane(myTable);

    //Add the scroll pane to this window.
    getContentPane().add(scrollPane, BorderLayout.CENTER);
  }

  /**Add a column to the table
   * @param columnName the name of the column to add
   * @param columnData the data for that column
   */
  public void addColumn(Object columnName, Vector columnData){
    tableModel.addColumn(columnName);
    int numRows = tableModel.getRowCount();
    for (int i = 0; i < (columnData.size()-numRows); i++)
        tableModel.addRow(new Vector());
    int column = tableModel.getColumnCount()-1;
    for (int i=0; i <columnData.size(); i++){
        Object value = columnData.elementAt(i);
        tableModel.setValueAt(value,i,column);
    }
  }

  /**Move a row in the table */
  public void moveRow(int start, int end, int to){
    tableModel.moveRow(start,end,to);
  }

  /**Get the number of rows in the table */
  public int getRowCount(){
    return tableModel.getRowCount();
  }

  /**Get the column displayed at a given row */
  public Column getColumn(int row){
    return (Column) tableModel.getValueAt(row,0);
  }

  /**Inner class for save as action*/
  private class SaveAsAction extends AbstractAction {
    public void actionPerformed(ActionEvent evt) {
        FileDialog fd = new FileDialog(new Frame(),
                                  "Save As...", FileDialog.SAVE );
        fd.show();
        if(fd.getDirectory() != null && fd.getFile() != null){
          myFilePath = new String(fd.getDirectory() + fd.getFile() );
          save();
        }
    }
```

```
    }

    private void save(){
      if (myFilePath == null){
          FileDialog fd = new FileDialog(new Frame(), "Save As...",
FileDialog.SAVE );
          fd.show();
          if(fd.getDirectory() != null && fd.getFile() != null)
            myFilePath = new String(fd.getDirectory() + fd.getFile());
      }
      if (myFilePath != null){
          PrintWriter out = null;

          try {
            out = new PrintWriter(new FileWriter(myFilePath));
            int numRows = tableModel.getRowCount();
            int numColumns = tableModel.getColumnCount();
            for (int i =0; i < numColumns; i++){
                out.print(tableModel.getColumnName(i));
                out.print(", ");
            }
            out.println();
            for (int i =0; i < numRows; i++){
                for (int j =0; j < numColumns; j++){
                  if (j==0)
                      out.print("\"");
                  out.print(tableModel.getValueAt(i,j));
                  if (j==0)
                      out.print("\"");
                  if (j < numColumns-1)
                      out.print(", ");
                }
                out.println();
            }
          } catch (IOException e) {
            System.err.println("Caught IOException: " + e.getMessage());
          } finally {
            if (out != null)
                out.close();
          }
      }

    }

}
```

REFERENCES

American Concrete Institute 209R-82 (1986) *Prediction of Creep , Shrinkage and temperature effects in Concrete structures*. ACI Manual of Concrete Practice, Michigan.

Beasley A.J. (1987) COLECS: *A Computer Program for the analysis of Elastic, Creep, Shrinkage and Thermal Deformations in the Columns and Cores of tall Buildings.* Research Report CM87/3, UTAS.

Deitel H.M., Deitel P.J. (1998) *JAVA How to Program.* Prentice Hall, USA.

Eckstein R., Loy M., Wood D. (1998) *JAVA Swing.* O Reilly & Associates, USA.

Jardin C.A., Dixon P. (1997) *Visual Caf Source Book.* John Wiley & Sons, Canada.

Knudsen J. (1999) *JAVA 2D Graphics.* O Reilly & Associates, USA.